TAMPEREEN TEKNILLINEN YLIOPISTO

JARI RAUHAMÄKI

UML-MALLIPOHJAISEN SOVELLUSKEHITYKSEN OHJEISTUS AUTOMAATIO-OHJELMISTOILLE

Diplomityö

Tarkastajat: professori Seppo Kuikka
tutkija David Hästbacka
Tarkastajat ja aihe hyväksytty
Automaatio-, kone- ja materiaalitekniikan tiedekuntaneuvoston
kokouksessa 4. maaliskuuta 2009

TIIVISTELMÄ

TAMPEREEN TEKNILLINEN YLIOPISTO
Automaatiotekniikan koulutusohjelma
RAUHAMÄKI, JARI: UML-mallipohjaisen sovelluskehityksen ohjeistus automaatio-ohjelmistoille
Diplomityö, 145 sivua, 18 liitesivua
Marraskuu 2009
Pääaine: Automaatioteknologia
Tarkastajat: professori Seppo Kuikka ja tutkija David Hästbacka
Avainsanat: Automaatiosuunnittelu, mallipohjainen kehitysprosessi, ohjeistus, UML-automaatioprofiili, automaatiosovellus

Automaatiosuunnittelu kohtaa jatkuvasti kasvavia haasteita kuten kiristyvän kilpailun ja tiukentuneet suunnitteluaikataulut. Suunnittelu on entistä enemmän ohjelmistopainotteista, koska ohjelmistojen osuus automaatiojärjestelmässä kasvaa jatkuvasti. Automaatiosuunnittelussa ei kuitenkaan hyödynnetä ohjelmistotekniikan uusia menetelmiä, joilla automaation ohjelmistosuunnittelua ja sen integroituvuutta muihin suunnitteluvaiheisiin voitaisiin tehostaa. AUKOTON-projektin tavoitteena on yhdistää tavanomaisessa ohjelmistotekniikassa hyödynnetyt menetelmät kuten UML ja mallipohjainen kehitysprosessi automaatio-ohjelmistojen suunnitteluun. Uuden automaatio-ohjelmistojen kehitysprosessin avulla pyritään tehostamaan ratkaisuiden uudelleenkäyttöä, parantamaan ohjelmistojen tuottamisen integroituvuutta muihin suunnitteluvaiheisiin ja mahdollistamaan sovelluskoodin automaattinen generointi sovelluksen mallin perusteella.

Tämän diplomityön aiheena on toteuttaa ohjeistus automaatio-ohjelmistojen toteuttamiseen suunnatun mallipohjaisen kehitysprosessin ja siinä hyödynnettävän, UML-automaatioprofiilissa määritellyn, käsitteistön käyttöön. Automaation ohjelmistosuunnittelussa ei yleensä ole hyödynnetty UML-kieleen pohjautuvaa mallinnuskäsitteistöä tai mallipohjaista kehitysprosessia, ei ainakaan sellaista, jollaista AUKOTON-projektissa kehitetään. Tästä syystä sekä kehitysprosessissa sovellettava käsitteistö, että itse kehitysprosessi vaativat ohjeistusta, jotta nykyiset automaatiosuunnittelijat, joilla ei välttämättä ole riittävästi kokemusta UML:n ja mallipohjaisten menetelmien soveltamisesta, voisivat hyödyntää kehitysprosessia ja sen käsitteistöä.

Työssä toteutettiin ja ideoitiin kahdenlaista ohjeistusta. Kirjallinen ohjeistus käsittelee automaatioprofiilia ja sen määrittelemiä käsitteitä sekä kehitysprosessia. Ohjeistuksessa tarjotaan esimerkkejä automaatioprofiilin käsitteistön käytöstä UML AP-työkalun yhteydessä, joka on kehitysprosessin keskeinen työkalu. Toisentyyppistä ohjeistuksen muotoa edustaa AP-työkaluun integroitu ohjeistus. Tämä ohjeistusmuoto käsittelee lähinnä kehitysprosessia tarjoten käyttäjälle ohjeita sen eri vaiheista ja tehtävistä. Integroitu ohjeistus avustaa käyttäjää nimenomaan AP-työkalun kontekstissa. Kirjallista ohjeistusta tullaan myös integroimaan AP-työkalun yhteyteen, jolloin se on helposti käyttäjän saatavilla.

Projektissa kehitettävä automaatio-ohjelmistojen mallipohjainen kehitysprosessi ja sen hyödyntämä UML-profiiliin perustuva käsitteistö on automaation suunnittelussa toistaiseksi harvinainen lähestymistapa suunnitteluun. Tässä diplomityössä kehitettävä ohjeistus maadoittaa uutta lähestymistapaa perinteisiin suunnittelumenetelmiin, kehittää sen käytettävyyttä ja osaltaan auttaa automaatiosuunnittelijoita siirtymään uuden kehitysprosessin käyttöön.

ABSTRACT

TAMPERE UNIVERSITY OF TECHNOLOGY
Master's Degree Programme in Automation Technology
RAUHAMÄKI, JARI: Guidance for UML based model-driven development of automation applications
Master of Science Thesis, 145 pages, 18 appendix pages
November 2009
Major: Automation technology
Examiners: Professor Seppo Kuikka and Research Scientist David Hästbacka
Keywords: Automation design, model-driven development process, guidance, UML Automation Profile, automation software

Design of automation systems faces ever increasing challenges such as tightening competition and schedules. The design process has become more and more software oriented. Yet the design process of automation software engineering does not utilize new development methods from software engineering, which could rationalize the automation software design and provide better integration for other design phases. The aim of the AUKOTON project is to combine UML and the model-driven development approach from common software engineering into the design process of automation software. The new development process of automation software aims to intensify the re-use of solutions, improve integration between design phases and enable automated code generation based on the model of the control application.

The subject of this thesis is to realize guidance for the model-driven development process of control software and the concepts utilized in the process, which are defined in the UML Automation Profile. UML based modeling concepts and model-driven development have not been utilized in automation software development, at least not a kind of process developed in the AUKOTON project. Thus the development process itself and the modeling concepts utilized in it require guidance because current automation engineers do not necessarily have enough experience in UML and model-driven methods to be able to utilize them.

Two kinds of guidance were developed and composed in the context of this thesis. The written user guide covers the UML Automation Profile, the concepts defined in it and the development process. The guide provides examples of the usage of the concepts of the Automation Profile in context of the UML AP Tool, which is a central tool in the development process. Guides and user assistance integrated into the AP Tool represents a different type of guidance. The integrated guidance mostly addresses the development process providing guidance for the tasks and the phases of the development process. The integrated guidance provides help particularly in the context of the AP Tool. However, the written guide will be integrated into the AP Tool so that the user has an easy access to it.

The model-driven development process for automation software developed in the project and the UML based modeling concepts utilized by the process is an exceptional approach to the design of automation systems for the time being. The guidance developed in this thesis is meant to ground the new approach to traditional design principles, to enhance its usability and to assist the automation engineers during the transition to the new development process.

ALKUSANAT

Tämä diplomityö on tehty Tampereen teknillisessä yliopistossa Systeemitekniikan laitokselle. Työ suoritettiin osana ”Automaatiosovellusten yhtenäinen kehityspolku PI-kaavioista ohjelmistotototeutukseen” (AUKOTON) -projektia. Diplomityö oli osa AUKOTON-projektin työpakettia, jonka tarkoituksena on toteuttaa ohjeistus kehityspolun ja siinä käytettävän toimialakohtaisen käsitteistön käytännön soveltamiseen sekä kehittää edelleen UML-automaatioprofiilissa määriteltyä toimialakohtaista käsitteistöä. Tämä diplomityö kattaa työpaketin tehtävistä ohjeistuksen toteuttamisen.

Haluan esittää kiitokseni työni ohjaajina ja tarkastajina toimineille professori Seppo Kuikalle ja tutkija David Hästbackalle työni tarkastamisesta sekä ohjeista ja tuesta, joita he minulle soivat työn tekemisen aikana. Lisäksi haluan kiittää mahdollisuudesta tehdä diplomityöni projektiin, joka toivoni mukaan avaa automaatio-ohjelmistojen suunnitteluun ja toteutukseen uusia mahdollisuuksia. Esitän kiitokseni myös muille tutkimusryhmäni jäsenille kommenteista ja neuvoista, erityisesti Timo Vepsäläiselle, joka kehityspolun työkalun kehittäjänä joutui (ja joutunee vastakin) sangen usein vastailemaan esittämiini työkalua ja kehityspolkua koskeviin kysymyksiin. Kiitän myös muita AUKOTON-projektityöryhmän jäseniä heidän panoksestaan ohjeistuksen ja tämän diplomityön toteuttamiseksi.

Lämpimät kiitokset kuuluvat myös rakkaalle vaimolleni Anne-Marille sekä vanhemmilleni saamastani tuesta ja kannustuksesta.

Tampereella 19.10.2009

_____

Jari Rauhamäki

SISÄLLYS

1. Johdanto 1
2. Automaatiosuunnittelun nykykäytännöt 3
2.1. Automaatiojärjestelmän suunnitteluprosessi 4
2.2. Määrittelyvaihe 5
2.2.1. Esisuunnittelu 6
2.2.2. Perussuunnittelu 7
2.3. Toteutussuunnitteluvaihe 7
2.3.1. Mekaniikkasuunnittelu 8
2.3.2. Laitteistosuunnittelu 8
2.3.3. Ohjelmistosuunnittelu 9
2.4. Nykyisten suunnittelukäytäntöjen kehityskohteita 10
3. Mallipohjaisesta ohjelmistokehityksestä 12
3.1. Yleistä 12
3.1.1. Lähestymistapoja mallien hyödyntämiseen ohjelmistotekniikassa 12
3.1.2. Hyötynäkökohtia 15
3.2. Model-Driven Architecture 17
3.2.1. MDA-mallien abstraktiotasot 17
3.2.2. MDA-prosessi 20
4. Mallipohjaisten automaatio-ohjelmistojen kehitysympäristö 22
4.1. Yleinen käsitteistö 22
4.1.1. Vaatimus 23
4.1.2. Automaatiotoiminto 23
4.1.3. Automaatiokomponentti ja -laite 24
4.2. UML 24
4.2.1. Metamallinnuksesta 25
4.2.2. UML-profiilimekanismi 27
4.3. UML-automaatioprofiili 28
4.3.1. Automaatioprofiilin arkkitehtuurista 29
4.3.2. Kaaviotyypit 31
4.3.3. Requirements-aliprofiili 32
4.3.4. AutomationConcepts-aliprofiili 37
4.3.5. DistributionAndConcurrency-aliprofiili 44
4.3.6. DevicesAndResources-aliprofiili 50
4.4. UML AP-työkalu 56
5. Automaatiosovelluksen UML-mallipohjainen kehitysprosessi 58
5.1. AUKOTON-lähestymistapa automaatiosuunnitteluun 58
5.2. Esimerkkiprosessi 60
5.3. Kehitysprosessin vaiheet 62
5.3.1. Edeltävät suunnitteluvaiheet 64

5.3.2. Lähtötietojen tuonti AP-työkaluun 68
5.3.3. Vaatimusten tarkentaminen 70
5.3.4. Vaatimuksista automaatiotoimintoihin 73
5.3.5. Automaatiotoimintojen tarkentaminen 74
5.3.6. Automaatiotoiminnoista automaatiokomponentteihin 79
5.3.7. Automaatiokomponenttien tarkentaminen 82
5.3.8. Automaatiokomponenteista alustakohtaiseksi suoritettavaksi sovellukseksi 87
5.3.9. Seuraavat työvaiheet 88
5.4. Mallipohjaisen lähestymistavan etuja ja haittoja automaatiosuunnittelun kannalta 89
6. Kehitysprosessin ohjeistus 91
6.1. Ohjeistuksen tarpeen perustelu ja kohderyhmä 91
6.2. Ohjeistuksesta yleisesti 92
6.3. Kirjallinen ohjeistus 93
6.4. Työkaluun integroidut ohjeet ja avusteet 95
6.4.1. Työkaluun integroitu kirjallinen ohjeistus 95
6.4.2. Cheat sheet 97
6.4.3. Wizard 99
6.4.4. Muita työkalualustan mahdollistamia ohjeistus- ja avustemuotoja 103
7. Johtopäätökset 104
7.1. Tekijän näkemyksiä ohjeistuksesta ja sen toteutuksesta 104
7.2. AUKOTON-projektityöryhmän jäsenten arvioita ohjeistuksesta 106
7.3. Automaatiosuunnittelijoiden arvioita ohjeistuksesta arviointitapahtuman pohjalta 107
7.4. Jatkokehitysideoita 109
8. Yhteenveto 112
Lähdeluettelo 114
Liite 1: Katkelma kirjallisesta ohjeistuksesta 119
Liite 2: AUKOTON-kehitysprosessia kuvaava cheat sheet 127
Liite 3: Control loop creator wizardin käyttötapaus 128

TERMIT JA NIIDEN MÄÄRITELMÄT

AUKOTON	AUKOTON on TTY:n, TKK:n ja VTT:n yhteishanke, jonka tarkoituksena on kehittää mallipohjainen automaatiosovellusten kehitysprosessi. Hanke päättyy vuoden 2009 loppuun.
Automaatiokomponentti	Automaatiosovellusten mallipohjaisen kehitysprosessin kontekstissa automaatiokomponentti edustaa ajoympäristössä suoritettavaa ohjelmakomponenttia, joka toteuttaa automaatiotoimintojen määrittelemän toiminnallisuuden.
Automaatiotoiminto	Automaatiosovellusten mallipohjaisen kehitysprosessin kontekstissa automaatiotoiminto edustaa automaatiojärjestelmän toimintoa tai toiminnallisuutta, kuten säätösilmukkaa tai säätöalgoritmia. Automaatiotoiminnoilla kuvataan järjestelmän looginen ja toiminnallinen malli.
CAEX	Computer Aided Engineering eXchange on neutraali dataformaatti hierarkkisen objektimuotoisen informaation tallentamiseen ja siirtämiseen. CAEX on XML-pohjainen ja se on määritelty XML-skeemana.
Cheat sheet	Eclipse-alustan tarjoama avuste- ja ohjeistusmekanismi, jonka avulla voidaan toteuttaa osin tai kokonaan automatisoituja askelohjeita esimerkiksi työnkulun läpikäymiseen.
CORBA	Common Object Request Broker Architecture on kommunikaatioväliohjelmisto, joka eristää kommunikaation yksityiskohdat sovellukselta ja sovellusohjelmoijalta.
DCS	Distributed Control System on säätöjärjestelmä, jossa säätöalgoritmien suoritus on hajautettu useille prosessointiyksiköille yhden keskitetyn yksikön sijasta.
EDDL	Electronic Device Description Language on IEC-standardi älykkäiden kenttälaitteiden kuvaamiseen.
EJB	Enterprise Java Beans on palvelinpuolen komponenttiarkkitehtuuri yritystason modulaaristen sovellusten toteuttamiseen.
FDT	Field Device Tools on standardi kenttälaitteiden välisiin sekä niiden ja isäntäjärjestelmän välisiin kommunikaatio- ja konfigurointirajapintoihin.
JRE	Java Runtime Environment on ohjelmisto, jonka avulla voidaan suorittaa Java-kielellä toteutettuja ohjelmia.
JSP	JavaServer Pages on Java-teknologia, jolla voidaan toteuttaa dynaamisesti generoitavia web-sivuja.
MDA	Model-Driven Architecture on OMG:n kehittämä malleja hyödyntävä ohjelmistokehityksen lähestymistapa. MDA:ssa

	ohjelmisto mallinnetaan ensin toteutusriippumattomalla käsitteistöllä, joka muunnetaan toteutusriippuvaksi ja lopulta suoritettavaksi sovellukseksi (osin) automaattisten muunnosten avulla.
MDD	Model-Driven Development on mallipohjainen lähestymistapa suunnitteluun, jossa pääasiallisena suunnitteluartefakteina ovat mallit dokumenttien sijaan.
MDE	Model-Driven Engineering viittaa mallipohjaiseen kehitysprosessiin ja suunnitteluun yleensä, eli se ei varsinaisesti ole rajoittunut ohjelmistokehitykseen vaan ottaa kantaa myös prosessiin ja määrittelyyn arkkitehtuurin ohessa
MDSD	Model-Driven Software Development on nimenomaan mallipohjaiseen ohjelmistokehitykseen suunnattu mallipohjainen menetelmä.
MOF	Meta-Object Facility on OMG:n standardoima metadatan määrittelykieli, jolla myös UML 2.0 on määritelty.
OMG	Object Management Group on vuonna 1989 perustettu avoin konsortio, joka on keskittynyt luomaan alustariippumattomia standardeja ohjelmistotekniikan alueelle. OMG:n ylläpitämiä standardeja ovat muun muassa UML, SysML, MDA ja CORBA.
PLC	Programmable Logic Controller on (yleensä pieni) tietokone, jota käytetään esimerkiksi automaatiossa suorittamaan ohjaus- tai säätöalgoritmia. Yleensä PLC viittaa erilliseen koteloituun tietokoneeseen, mutta se voi viitata myös PC:ssä ajettavaan PLC-ohjelmistoon, tällöin kuitenkin yleensä puhutaan Soft-PLC:stä.
QoS-FT-profiili	Quality of Service and Fault Tolerance -profile on UML:n laajennus (profiili), joka sisältää käsitteistöä palvelunlaadun ja vikasietoisuuden mallintamiseen.
RT-profiili	UML-profiili ajoitettavuudelle, suorituskyvylle ja aikamääritteille on UML:n laajennus (profiili), joka sisältää käsitteistöä ajoitettavuuden, suorituskyvyn ja aikamääritteiden mallintamiseen.
SQL	Structured Query Language on standardoitu kyselykieli, jolla voidaan tehdä hakuja relaatiotietokantoihin.
Stereotyyppi	Stereotyyppi on yksi UML:n laajennusmekanismeista, jolla käyttäjä voi luoda uusia mallinnuselementtejä ja näin laajentaa UML:n mallinnuskäsitteistöä. Stereotyyppi erikoistaa jo olemassa olevaa UML:n elementtiä antaen sille uuden/tarkennetun merkityksen.

SysML	System Modeling Language on UML:n laajennus (profiili), joka sisältää käsitteistöä systeemien mallintamiseen.
Toimilohkotyyppi	Toimilohkotyyppi on automaatiosuunnittelussa käytetty artefakti, joka edustaa jotain tiettyä automaatiojärjestelmän toiminnallisuutta, esimerkiksi säätöalgoritmia. Toimilohkotyyppi sisältää parametreja, jotka määrittelevät sen toimintaa sekä tuloja ja lähtöjä, joilla se voidaan kytkeä muihin toimilohkotyyppeihin.
UML	Unified Modeling Language on OMG:n vuonna 1997 standardoima mallinnuskieli, joka on alun perin tarkoitettu ohjelmistojen kehityksen ja suunnittelun tukemiseen, mutta on laajennettavissa myös muille toimialoille sopivaksi. Laajennusmahdollisuus, eli UML-profiilit, määriteltiin alun perin UML:n versioon 1, mutta UML:n versiossa 2.0 profiilien tukea parannettiin merkittävästi. UML on nykyisin versiossa 2.2.
UML AP-työkalu	Eclipse-työkalualustan perustalle kehitetty UML-mallinnustyökalu, joka tukee automaatioprofiilin käsitteistöä ja AUKOTON-kehitysprosessia.
UML-automaatioprofiili	USVA-hankkeessa määritelty ja myöhemmissä projekteissa, muun muassa AUKOTON-projektissa, täydennetty UML-profiili, joka sisältää automaatioalalle tyypillistä käsitteistöä.
Wizard	Käyttäjää avustava toiminto, joka helpottaa monimutkaisten toimintojen suorittamista. Wizard pyytää käyttäjältä tietoja ja niiden perusteella toteuttaa toiminnon, esimerkiksi luo uuden projektin tarvittavine kansioineen ja tiedostoineen.
XML	eXtensible Markup Language on rakenteisten dokumenttien merkintäkieli, jolla voidaan kuvata sekä tietoa itseään, että sen rakennetta.

1. JOHDANTO

Automaatioalalla kilpailu on kiristänyt tuotantoaikatauluja, jolloin myös automaatio-ohjelmistojen kehitysaikoja on jouduttu leikkaamaan. Toisaalta kehitettävistä järjestelmistä on tullut jatkuvasti monimutkaisempia ja siten haastavampia ohjelmistoteknisesti. Nämä seikat ovat asettaneet paineita automaation ohjelmistokehitysprosessin tehostamiselle. Automaatio-ohjelmistojen kehitysprosessi on kuitenkin perinteisesti ollut hajanainen. Kehitysprosessiin osallistuvat tahot ovat toimineet omien käsitteiden, työkalujen ja tapojensa mukaan. Yhteistyö eri tahojen välillä on ollut vaikeaa.

AUKOTON-projektissa kehitettävä mallipohjainen ohjelmistokehitysprosessi pyrkii vastaamaan esitettyyn tehostamistarpeeseen. Projektin tarkoituksena on luoda yhtenäinen automaatio-ohjelmistojen kehitysympäristö, jossa eri suunnittelutahot voivat toimia samojen käsitteiden, työkalujen ja tapojen mukaan. Ympäristössä luodaan edellytykset muun muassa korkeantason mallintamiselle, uudelleenkäytettävyyden hyödyntämiselle ja manuaalisen tiedonsiirron minimoimiselle. Näin saadaan rajalliset suunnitteluresurssit valjastettua tuottavan suunnittelutyön käyttöön, eikä resursseja huku manuaaliseen tiedonsiirtoon ja suunnitteluosapuolten välisistä eriävistä käsitteistä aiheutuvien virheiden korjaamiseen.

Uusi mallipohjainen kehitysprosessi perustuu standardin mallinnuskielen käyttöön läpi koko automaatio-ohjelmiston kehitysprosessin. UML tarjoaa monipuolisen ja laajennettavan mallinnuskäsitteistön ja kuvaustyypit, jotka ovat jo yleisesti käytössä esimerkiksi ohjelmistotekniikan alueella. Standardi UML-käsitteistö ei kuitenkaan ole sellaisenaan riittävä automaatiosovellusten kuvaamiseen. USVA-projektissa kehitetty UML-automaatioprofiili määrittelee automaatioalan käsitteitä ja kuvaustyyppejä, joilla automaatio-ohjelmiston mallinnusta, standardi UML-käsitteistön tuella, voidaan toteuttaa. Automaatioprofiilissa määritellyt käsitteet ja kuvaustyypit ovat käytettävissä UML AP-työkalussa, jolla automaatio-ohjelmisto voidaan suunnitella yhtenäisenä ketjuna aina vaatimuksista suoritettavaan ohjelmaan asti. Automaatiosuunnittelussa ei kuitenkaan perinteisesti ole käytetty UML:n ja mallipohjaisen kehityksen kaltaisia menetelmiä. Näistä syistä uuden kehitysympäristön käyttöön tarvitaan ohjeistusta, jotta se voitaisiin saattaa suunnittelijoiden käytettäväksi ja hyväksyä alalla yleisemminkin.

Tämän diplomityön tarkoituksena on ollut kehittää kattava ja yksikäsitteinen ohjeistus kehitysympäristön ja AUKOTON-kehitysprosessin mukaisen työnkulun käyttöön. Tässä diplomityössä ei oteta kantaa ohjelmistokehityksen ulkopuolisiin asioihin kuten prosessisuunnitteluun. Ohjeistus on kohdistettu automaatiosuunnittelijoille ja siitä on pyritty saamaan esitystavaltaan sellainen, että myös UML:ää ennestään tuntematon henkilö pystyy ohjeistuksen avulla käyttämään kehitysympäristöä, eli automaatioprofiilia ja

UML-automaatioprofiilityökalua. Lisäksi on pyritty huomioimaan eri käyttäjäryhmien, eli suunnitteluun osallistuvien tahojen, erilaiset taustat ja saattamaan ohjeistus kaikkien ryhmien ymmärtämään muotoon. Ohjeistuksessa ei käydä läpi UML:n standardiversion ominaisuuksia, notaatiota ja kuvaustyyppejä muuta kuin niiltä osin, joissa se on katsottu kehitysympäristön käytön kannalta oleelliseksi.

Ohjeistuksen runko on tehty UML-automaatioprofiilin ympärille esittelemällä sen mallinnuskonsepteja ja kaaviotyyppejä. Toinen tärkeä kokonaisuus on mallipohjainen kehitysprosessi ja sen toteutustapa AUKOTON-projektin konseptein. Ohjeistus on toteutettu sekä perinteisenä ohjeena, että UML AP-työkaluun integroituina interaktiivisina ja sulautettuina opasteina sekä ohjeina.

Vaikka diplomityön pääasiallinen tavoite on ollut toteuttaa ohjeistus AUKOTON-projektin puitteissa, voidaan tällä diplomityöllä nähdä myös toisenlainen merkitys. Diplomityö sisältää varsin laajan esityksen AUKOTON-projektin taustoista, periaatteista sekä sovellettavista käsitteistä, tekniikoista ja menetelmistä. Tästä syystä tätä diplomityötä voidaan pitää hyvänä AUKOTON-projektin tuotoksia esittelevänä yleisteoksena, jonka avulla lukija voi aloittaa tutustumisen mallipohjaiseen automaatio-ohjelmistojen kehitysprosessiin.

Diplomityö etenee automaatiosuunnittelun nykykäytännöistä mallipohjaisen automaatio-ohjelmistojen kehitysprosessin kuvaamiseen ja lopulta kehitysprosessille tehdyn ohjeistuksen esittelyyn. Diplomityön luvussa kaksi tarkastellaan automaatiosuunnittelun nykykäytäntöjä ja niiden kehityskohteita. Luvussa kolme esitellään mallipohjaista ohjelmistokehitystä yleisesti, jotta lukija saa yleiskuvan niistä tekniikoista ja menetelmistä, joihin mallipohjainen automaatio-ohjelmistojen kehitys perustuu. Mallipohjainen kehitys mahdollistaa ja vaatii toimialakohtaisen mallinnuskäsitteistön hyödyntämistä ja työkalutukea mallinnuskäsitteistön käyttämiselle. Luvussa neljä esitelläänkin AUKOTON-kehitysprosessin mallinnuskäsitteistö, joka on määritelty UML-automaatioprofiilissa, sekä UML AP-työkalu, joka tukee automaatioprofiilin käsitteistöä ohjelmiston mallinnuksessa. Luku viisi kuvaa kehitetyn automaatio-ohjelmistojen mallipohjaisen kehitysprosessin käyttäjälähtöisesti ja esittää käytännössä miten AP-työkalua ja automaatioprofiilin käsitteitä hyödynnetään järjestelmän mallintamisessa. Luvussa kuusi esitellään kehitysprosessille toteutettua varsinaista ohjeistusta. Luvussa seitsemän pohditaan ja arvioidaan diplomityötä sekä esitetään työlle jatkokehitysideoita.

2. AUTOMAATIOSUUNNITTELUN NYKYKÄYTÄNNÖT

Automaatiolla tarkoitetaan yleensä teollisen prosessin hallintaa ja säätöä automaattisin keinoin, ilman ihmisen merkittävää myötävaikutusta [1]. Ihmisen myötävaikutusta tarvitaan prosessin tarkkailussa, mutta tarkoitus on, että ihmisen myötävaikutus on minimaalista laitteiston toiminnan kannalta. Itseohjautuvuudessa voidaan nähdä useita tasoja, mutta yleisimmin kyse on suureen pitäminen tietyssä arvossa, kuten säiliössä olevan nesteen pinnankorkeuden vakioiminen erilaisten pinnankorkeuteen vaikuttavien häiriöiden läsnä ollessa. Toisaalta jopa kokonaisten tuotantolaitosten itseohjautuvuus on saavutettavissa. Automaatiojärjestelmänä voidaan pitää laitteistoa ja siihen (mahdollisesti) liitettyä ohjelmistoa, jotka toteuttavat määritellyn itseohjautuvuuden.

Automaation merkitys teollisuudessa, erityisesti länsimaisessa, on jatkuvasti kasvussa. Jatkuvaa tuotantotehokkuuden kasvua ei mitenkään voida jatkuvasti saavuttaa ilman automaatiotason nostoa varsinkaan Suomessa, missä työvoiman kapasiteetilla ja palkkatasolla ei pystytä kilpailemaan teollisessa tuotannossa. Tehostamistarve ei kuitenkaan rajoitu pelkästään automaation käyttöön ja sen suoriutumistasoon, vaan paineita tehokkuuden nostamiselle on myös automaatiosuunnittelun kuten muidenkin suunnittelualojen alueella. Suunniteltavat järjestelmät ovat jatkuvasti monimutkaisempia ja laajempia. Samalla kuitenkin kiristyneen kilpailun takia suunnitteluprojektit viedään läpi entistä kireämmillä aikatauluilla, mikä väistämättä pakottaa nostamaan suunnittelun tehokkuutta.

Automaatiosuunnittelun tehtävä on kehittää automaatiojärjestelmä, joka pystyy toteuttamaan siltä vaaditun toiminnallisuuden. Automaatiosuunnittelu on tavallisesti osa laajempaa kokonaisuutta, esimerkiksi uuden laitoksen suunnittelua tai jo olemassa olevan prosessin modernisointia. Halmetojan mukaan automaatiosuunnittelun voidaan katsoa kattavan laajemmasta kokonaisuudesta seuraavat osa-alueet: automaation instrumentoinnin, sähköistyksen, laitteiston ja automaatiosovellusten suunnittelun [2]. Näistä osatekijöistä muodostuu suunnittelun lopputuloksena automaatiojärjestelmä.

Projektina automaatiosuunnittelu ei juuri eroa muista (vastaavan kokoluokan) projekteista. Suunnittelu lähtee liikkeelle vaatimuksista ja päätyy järjestelmän toteutukseen. Suunnittelusta voidaan erottaa karkeasti kolme päävaihetta: esisuunnittelu, perussuunnittelu ja toteutussuunnitteluvaiheet [3; 4]. Esi- ja perussuunnitteluvaiheet usein mielletään määrittelyvaiheeksi, jota seuraa varsinainen suunnitteluvaihe. Vaiheisiin tutustutaan tarkemmin myöhemmin tässä luvussa. Näin saadaan vertailupohjaa luvussa viisi esitettävälle automaatiosovellusten mallipohjaiselle kehitysprosessille. Halmetoja toteaa, että suunnitteluprosessin vaiheiden rajat ovat varsin sumeat, eikä niitä voida täysin

erottaa toisistaan. Suunnitteluprosessi on iteratiivinen, asteittain tarkentuva ja eri osatehtävät tuottavat siinä jatkuvasti tarkentuvaa tietoa toisillensa lähtötiedoiksi. [2]

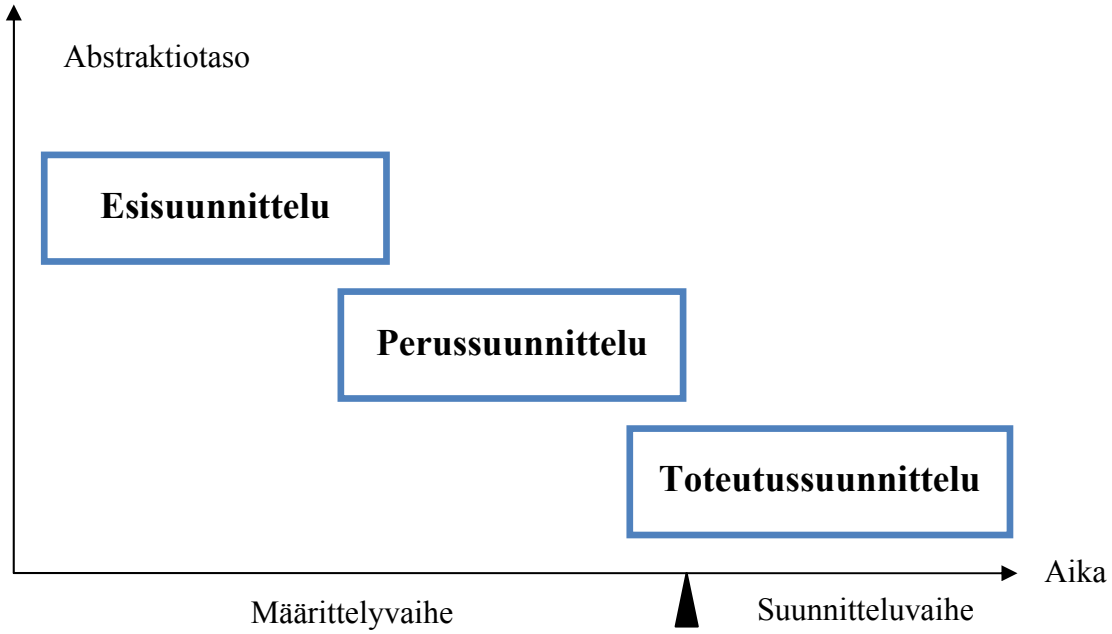
Judénin mukaan automaatiojärjestelmien tuotanto, niin laitteistojen kuin ohjelmistojen osalta, on nykyisin verkottunutta. Automaatiojärjestelmän suunnittelu siis hajautuu useille eri osapuolille, jotka toimittavat osan automaatiojärjestelmästä. Suunnitteluosapuolilla on omat kapeahkot osaamisalueensa, jolla ne toimivat. Yhdellä yrityksellä voi olla osaamista esimerkiksi prosessisuunnittelusta ja perussuunnittelusta, toisella instrumentoinnista ja kolmannella automaatiosovelluksista [3]. Normaalisti automaation suunnitteluprojektissa on mukana ainakin seuraavat osapuolet: asiakas, automaatiojärjestelmätoimittaja, prosessisuunnittelu, laitesuunnittelu ja sähkösuunnittelu [4]. Koska projektissa voi olla mukana suuri joukko eri suunnitteluosapuolia, nousee projektissa käytettävien työkalujen, menetelmien ja käsitteistöjen määrä suureksi. Viinikkala toteaakin, että suureksi haasteeksi projektissa nousee kommunikaatio ja tiedonsiirto eri suunnitteluvaiheiden, -osapuolten ja -työkalujen välillä. Projektin onnistumisen kannalta tiedon virheetöntä ja oikea-aikaista kulkua voidaankin pitää perustavanlaatuisena tekijänä. [5]

Tässä luvussa käydään läpi automaatiosuunnittelun nykykäytäntöjä. Automaation suunnitteluprosessia tarkastellaan ensin yleisesti, jonka jälkeen paneudutaan tarkemmin automaatiosovellusten suunnitteluun ja integroitumiseen osaksi automaatiosuunnittelua. On tärkeä huomata, että automaation suunnittelukäytännöt eroavat merkittävästi eri yritysten jopa yhden yrityksen eri projektien välillä [6]. Esitetty suunnitteluprosessi ei näin ollen ole välttämättä kaikissa automaatiosuunnitteluprojekteissa käytettävän kaltainen. Se pyrkii kuitenkin selventämään automaatiosuunnitteluprojektin yleisiä vaiheita.

2.1. Automaatiojärjestelmän suunnitteluprosessi

Automaatiosuunnittelu voidaan, kuten edellä mainittiin, jakaa karkeasti kolmeen vaiheeseen. Vaiheita ei voida pitää erillisinä, vaan ne etenevät osin rinnakkain ja iteratiivisesti. Tästä syystä esimerkiksi toisen osapuolen esisuunnittelu on lähtö- tai lisätietoa toisen osapuolen perussuunnitteluun, mikä hyvin ilmentää sitä, että eri suunnitteluosapuolet ja -vaiheet tuottavat toisilleen lähtötietoa tarkempaa suunnittelua varten. Juuri tästä syystä saumattoman tiedonkulun merkitys automaatioprojektissa korostuu, mutta toisaalta sitä on nykyisillä menettelytavoilla hankala saavuttaa. [4]

Kuvassa 2.1 on esitetty automaatiosuunnittelun päävaiheet ja suunnittelun viitteellinen eteneminen, koska, kuten edellä mainittiin, suunnitteluprosessi ei ole suoraviivainen vaan siinä on havaittavissa rinnakkaisuutta ja iteratiivisuutta. Suunnittelu kulkee esisuunnittelusta perussuunnittelun kautta varsinaiseen toteutussuunnitteluvaiheeseen, kun suunnittelua kuvataan ajassa etenevänä prosessina. Seuraavissa aliluvuissa tarkastellaan tarkemmin kutakin vaihetta.



Kuva 2.1. *Automaatiosuunnittelun päävaiheet ja eteneminen (mukailtu lähteestä [4])*

2.2. Määrittelyvaihe

Ensimmäinen vaihe automaation suunnitteluprosessissa on määrittely. Määrittelyvaiheen pääasiallinen tavoite on tuottaa riittävät lähtötiedot tarkempaa suunnittelua ja toteutusta varten. Näihin lähtötietoihin kuuluvat muun muassa automaatiojärjestelmän toiminnallisuus ja toiminnan reunaehdot, joita määrittävät esimerkiksi järjestelmän toivottu suorituskyky ja luotettavuus sekä paikalliset viranomaismääräykset. Lisäksi määrittelyvaiheessa sovitaan myös erilaisista käytännöistä ja vastuualuista projektin sisällä. [4]

Määrittelyvaihetta pidetään kriittisenä osana suunnittelua, koska suurin osa myöhemmin kohdattavista ongelmista on peräisin juuri määrittelyvaiheen puutteista ja virheistä [7, katso 4]. Perinteisen ohjelmistotekniikan puolella määrittelyvaiheen kriittisyys on havaittu jo kauan sitten, mutta automaation suunnittelussa ja ohjelmistosuunnittelussa vaatimuksien merkitystä voitaisiin painottaa nykyistä enemmänkin. [4]

Määrittelyvaihe voidaan karkeasti jakaa kolmeen alakohtaan, jotka ovat esitutkimus, esisuunnittelu ja perussuunnittelu. Esitutkimusta voidaan pitää esisuunnittelun ensimmäisenä iteraatiokierroksena. Esitutkimuksessa tutkitaan automaatiohankkeen toteuttamisen vaikutuksia ja sen tulosten perusteella päätetään, jatketaanko suunnitteluprosessia esisuunnitteluun [3; 4]. Esi- ja perussuunnitteluvaiheisiin tutustutaan tarkemmin seuraavissa aliluvuissa.

2.2.1. Esisuunnittelu

Kun esitutkimuksen perusteella, tai ilman sitä, on päätetty aloittaa varsinainen automaatiosuunnittelu, sen ensimmäinen vaihe on esisuunnittelu. Esisuunnitteluvaiheen tavoitteena on määritellä automaatiojärjestelmän vaatimukset ja niiden reunaehdot. Lisäksi tuotetaan arvio automaatiojärjestelmän hyödyistä ja kustannuksista, joiden perusteella voidaan tehdä investointipäätös. Investointipäätös on esi- ja perussuunnitteluvaiheita erottava etappi. [4]

Esisuunnittelu on pääasiassa asiakkaan vastuulla oleva tehtävä, mutta jos automaatiojärjestelmän toimittaja on jo valittu, voi asiakas tehdä esisuunnittelun yhdessä toimittajan kanssa. Tämä ei välttämättä ole pelkästään hyvä asia, sillä näin meneteltäessä toimittajan oma tai sen hyödyntämä laitteisto sen tarjoamine ominaisuuksineen ja rajoitteineen saatetaan ottaa vaatimusten lähtökohdaksi. Tässä vaiheessa suunnitteluprosessia esitettävien vaatimusten ei tulisi kohdistua laitteistoon tai automaatiojärjestelmän ajoalustaan. Sen sijaan vaatimusten tulisi ilmentää järjestelmän vaatimuksia käyttäjien, tuotannon ja ohjattavan järjestelmän toiminnan ja ominaisuuksien näkökulmista [4]. Toisaalta järjestelmätoimittajan asiantuntemus voi auttaa automaatiojärjestelmiä mahdollisesti tuntematonta asiakasta niin, että vaatimusmäärittelyyn saadaan koostettua vaiheen kannalta oleelliset seikat.

Vaatimuksia on kahta pääkategoriaa: toiminnallisia ja ei-toiminnallisia. Toiminnalliset vaatimukset määrittelevät mitä järjestelmän tulee tehdä, esimerkiksi: ”Jäähdytysvesitankin pinnankorkeuden tulee olla automaattisesti vakavoitu.”. Ei-toiminnalliset vaatimukset puolestaan määrittelevät mitä rajoitteita ja laadullisia piirteitä toiminnallisuuksilla on, eli ne ovat reunaehtoja suunnittelulle. Edelliseen esimerkkiin liittyvä ei-toiminnallinen vaatimus voisi olla esimerkiksi: ”Jäähdytysvesitankin pinnankorkeuden tulee pysyä +-100 millimetrin etäisyydellä asetusarvosta häiriöiden läsnä ollessa.”. Asmala et al. toteavat, että esisuunnitteluvaiheessa vaatimuksiin joudutaan poimimaan kohtia myös muiden suunnittelualojen dokumenteista. Esimerkiksi mekaniikkasuunnittelusta saattaa löytyä sellaisia tietoja, jotka merkittävästi vaikuttavat automaatiosuunnitteluun. [4]

Esisuunnittelun lopputulokset, eli automaatiojärjestelmän vaatimukset, dokumentoidaan. Esisuunnittelussa tulee esille monia sellaisia seikkoja, jotka vaikuttavat myös sovellussuunnitteluun, kuten tietenkin kaikkiin muihinkin suunnittelualueisiin. Tästä syystä olisi eduksi, että vaatimukset pystyttäisiin sähköisesti siirtämään eri suunnittelujärjestelmiin. Nykyinen käytäntö kuitenkin on, että vaatimukset esitetään perinteisinä tekstimuotoisina dokumentteina [2]. Vaikka tämä tapa ei olekaan paras mahdollinen manuaalisen työn minimoimiseksi vaatimusten siirtovaiheessa, on sen käytöllä perusteensa. Ensinnäkin tekstimuotoinen dokumentti on asiakkaan kannalta selkein tapa esittää vaatimukset ja toisekseen esisuunnittelun aikana tuotetaan vasta alustavat vaatimukset, joita edelleen tarkennetaan ainakin perussuunnitteluvaiheessa [8]. Toisaalta jos vaatimukset esitettäisiin standardisoiduilla yhteisillä käsitteillä, niitä voitaisiin hyödyntää laajalti erilaisissa työkaluissa, esimerkiksi automaatio-ohjelmistojen suunnittelutyökaluissa.

Tällöin muun muassa jäljitettävyyden säilyttäminen vaatimuksista ne toteuttaviin automaatiotoimintoihin ja sovelluskomponentteihin voitaisiin saavuttaa.

2.2.2. Perussuunnittelu

Esisuunnitteluvaiheen jälkeen automaatiojärjestelmän suunnitteluprosessissa siirrytään perussuunnitteluun. Näitä vaiheita erottaa asiakkaan investointipäätös. Investointipäätöksen jälkeen asiakas laatii esisuunnitteluvaiheessa määriteltyjen vaatimusten pohjalta tarjouspyynnön, johon automaatiotoimittajat vastaavat omalla näkemyksellään ratkaisusta. Perussuunnittelussa päävastuu on siis automaatiotoimittajilla, mutta yhteistyö asiakkaan ja muiden suunnitteluosapuolten kanssa on vielä tässä vaiheessa voimakasta. [2; 4]

Perussuunnittelun tavoite on tarkentaa esisuunnitteluvaiheen vaatimukset automaatiojärjestelmän toiminnalliseksi kuvaukseksi. Toiminnallinen kuvaus on suunnittelutoimiston tai automaatiotoimittajan ratkaisu asiakkaan esittämien vaatimusten määrittelemälle järjestelmälle. Se sisältää järjestelmän toimintojen kuvauksen laajoista kokonaisuuksista yksittäisiin automaatiotoimintoihin, liittymät käyttäjiin, laitteisiin ja toisiin järjestelmiin sekä esityksen ei-toiminnallisten vaatimusten toteuttamisesta. Toiminnallinen kuvaus ottaa yleensä kantaa myös järjestelmän toteutustekniikkaan esimerkiksi toimittajan laitteiston ja vakioratkaisuiden muodossa. [4; 8]

Keskeinen osa perussuunnittelun läpivientiä ovat eri suunnitteluosapuolten väliset keskustelut. Tässäkin vaiheessa siis korostuu eri osapuolten kommunikaatio suunnitteluprosessin edetessä. Keskusteluita järjestelmästä käydään muun muassa prosessi-, mekaniikka-, sähkö- ja hydrauliikkasuunnittelun kanssa. Erityisen oleellinen osa perussuunnittelua ovat ajotapakeskustelut. Ajotapakeskusteluissa tarkennetaan suunniteltavan prosessin pää- ja aputoiminnot ja tarkennetaan toimintojen yksityiskohtia. Ajotapa- ja muiden keskusteluiden ohella esisuunnitteluvaiheen vaatimukset tarkentuvat ja niiden pohjalta muodostuu automaatiojärjestelmän toiminnallinen kuvaus. [4; 9]

Perussuunnittelun lopputuloksena tulisi olla virheetön, täsmällinen, ristiriidaton ja todennettava suunniteltavan automaatiojärjestelmän toiminnallinen kuvaus. Kuvauksen tulisi olla yksiselitteinen, mutta toisaalta esitettynä sellaisessa muodossa, että sekä asiakkaan edustajat että toimittajan suunnittelijat pystyvät hyödyntämään sitä. Ongelmana onkin, että asiakkaan kannalta kuvauksen tulisi olla yksinkertainen, mutta toimittajan pitäisi pystyä hyödyntämään kuvausta suunnittelun pohjana. [8]

2.3. Toteutussuunnitteluvaihe

Suunnitteluvaihe käynnistyy kun asiakas perussuunnitteluvaiheen toimintakuvausten ja neuvotteluiden jälkeen valitsee automaatiojärjestelmän toimittajan (ellei valittu jo aiemmin) ja tarvittavat sopimukset allekirjoitetaan. Sopimusten allekirjoitus käynnistää toimittajan suunnitteluprosessin ja suunnitteluvastuu siirtyy lähes kokonaan toimittajalle. Yhteys asiakkaan ja toimittajan välillä kuitenkin säilyy esimerkiksi tarkentavien ajotapakeskusteluiden muodossa. Suunnittelun tarkoituksena on tarkentaa määrittelyvai-

heen tuloksien pohjalta automaatiojärjestelmän suunnitelmat sellaiselle tasolle, että niiden perusteella automaatiojärjestelmä voidaan toteuttaa. Suunnittelussa pyritään siis löytämään ratkaisut toiminnallisessa kuvauksessa esitetyn järjestelmän toteuttamiseksi. [8]

Suunnitteluvaihe voidaan jakaa karkeasti kolmeen vaiheeseen: mekaniikka-, laitteisto- ja ohjelmistosuunnitteluun. Suunnitteluvaiheen lähtökohtana ovat prosessilaitteet ja koneet, joiden jälkeen suunnitellaan laitteisto, sijoittelu, verkot ja ohjelmistot. Jälleen kerran eri vaiheet eivät etene peräkkäisinä vaiheina vaan iteratiivisesti ja rinnakkain toistensa kanssa. [8]. Mekaniikka-, laitteisto- ja ohjelmistosuunnittelua tarkastellaan tarkemmin seuraavissa aliotsakkeissa.

2.3.1. Mekaniikkasuunnittelu

Mekaniikkasuunnittelu on yleisesti prosessilaitteiden ja koneiden suunnittelua. Mekaniikkasuunnitteluun voi kuulua muun muassa putkisto- ja laitossuunnittelu prosessiteollisuuden alueella ja kappalekäsittelijän tai pakkaajan suunnittelua kappaletavarateollisuuden alueella. Mekaniikkasuunnittelu ei itse asiassa yleensä kuulu automaatiosuunnitteluun, mutta esimerkiksi pienimuotoisissa laitoksien tai niiden osien uudistuksissa myös mekaniikkasuunnittelua saatetaan asettaa automaatiosuunnittelun vastuulle. [5; 8]

Mekaniikkasuunnittelu on, vaikka ei yleensä automaatiosuunnittelua olekaan, tärkeä pohja sen läpiviemiseksi. Ilman esimerkiksi automatisoitavan prosessin putkisto- ja virtauskaavioita automaatiosuunnittelua ei pystytä ainakaan optimaalisella tasolla suorittamaan.

2.3.2. Laitteistosuunnittelu

Ajon et al. mukaan laitteistosuunnitteluvaihe alkaa, kun mekaniikkasuunnittelu on edennyt riittävän stabiiliin vaiheeseen. Laitteistosuunnittelussa täydennetään järjestelmän rakennetta, laitteistoa ja niiden sijoittelua automaatiojärjestelmän kannalta. Tässä vaiheessa siis määritellään muun muassa ohjaimet (esimerkiksi PLC- ja DCS-prosessiasemat), niiden liityntäkortit, yksikkösäätimet, PC:t ja sulautetut järjestelmät sekä instrumentointi ja kaapelointi. Lisäksi suunnitellaan operointiin ja huoltoon liittyvät käyttöliittymät. [8]. Laitteistosuunnittelussa siis suunnitellaan automaatiojärjestelmän laitteisto. Jos automaatiojärjestelmä on hajautettu, kuuluu laitteistosuunnitteluun olennaisena osana myös verkkosuunnittelu. Siinä suunnitellaan verkon fyysiset komponentit kuten reitittimet, palvelimet ja kaapelointi. [5; 8]

Laitteistosuunnittelu on sidoksissa erityisesti mekaniikkasuunnitteluun, koska automaatiolaitteisto on voimakkaasti riippuvainen prosessissa käytettävistä mekaniikkaratkaisuista. Pienikin muutos mekaniikassa voi tästä syystä aiheuttaa suuria muutoksia automaatiolaitteistossa [8]. Laitteistosuunnittelun lopputuloksena on laitteistokuvaus, joka kuvaa sen laitteiston, jolla automaatiosovellusta suoritetaan ja jota se ohjaa [4]. Kun laitteisto on saatu suunniteltua, se tilataan joko erilliseltä valmistajalta tai automaatiojärjestelmätoimittajan laitevalmistusosastolta.

2.3.3. Ohjelmistosuunnittelu

Ohjelmistosuunnittelu etenee rinnan laitteistosuunnittelun kanssa. Se saa lähtötietoja toiminnallisen kuvauksen ohella myös muun muassa laitteistokuvauksista. Suunnittelu perustuu näin ollen tarkoille tiedoille käytettävistä instrumenteista, moottoreista, pumpuista ja niin edelleen [8]. Usein suunnittelu toteutetaan samoilla artefakteilla kuin varsinainen toteutuskin, eli esimerkiksi tietyn DCS-järjestelmän alustariippuvilla tyyppipiireillä. Tämä yhdessä suunnittelun laitteistosidonnaisuuden kanssa antaa ohjelmistosuunnittelulle tyypillisesti voimakkaan alustariippuvuuden jo sen alkuvaiheista asti. Tämänkaltainen alustariippuva suunnittelu riistää ratkaisuiden uudelleenkäyttömahdollisuuksia toisilla alustoilla, joille sama ratkaisu joudutaan kehittämään erikseen sitä tarvittaessa. Ratkaisun idea ja konsepti voidaan tietenkin (suunnittelijan ajatuksina) siirtää eri järjestelmille, mutta tämä ei ole järin tehokasta ratkaisuiden uudelleenkäyttöä.

Ohjelmistosuunnittelussa tulisi käyttää [8] ja myös käytetään [5] mahdollisimman paljon valmiita, testattuja ja hyväksyttyjä ohjelmistokomponentteja, joita automaatioalalla usein ilmentävät tyyppipiirit. Tyyppipiirit ovat itse asiassa yksi syy ohjelmistosuunnittelun alustasuuntautuneisuuteen automaatioalalla. Yleensä tyyppipiirit ovat tietylle alustalle määriteltyjä laitteiden ja toimintojen malleja, joilla voidaan kyseiselle alustalle suunnitella ja toteuttaa toimiva kokonaisuus. Tämä antaa tyyppipiireille alustariippuvan leiman. Tyyppipiirejä voidaan kuitenkin toteuttaa myös niin, että ne kuvaavat järjestelmää alustariippumattomasti. Tällaisia tyyppipiirejä voidaan käyttää esimerkiksi suunniteltaessa järjestelmiä, joita käytetään eri kohdealustoilla. Alustariippumattomia tyyppipiirejä voidaankin tästä syystä käyttää esimerkiksi automaation perussuunnittelussa, jossa ei oteta merkittävästi kantaa siihen laitteistoon ja ohjelmistoon, joilla lopullinen automaatiojärjestelmä toteutetaan.

Automaatiosuunnittelussa ohjelmistosuunnitteluvaihe nivoutuu varsin tiiviisti toteutusvaiheeseen. Suunnittelussa hyödynnetään toteutuskomponentteja, joten tiukkaa rajanvetoa ohjelmistosuunnittelun ja toteutuksen välillä ei voida tehdä, vaan jossain vaiheessa suunnittelusta siirrytään saumattomasti toteutukseen. Ohjelmiston rakenne, tietokannat, tyyppipiirit yms. suunnitellaan yleensä rajapinta- ja liityntätasolla. Näistä muodostuu ohjelmiston runko, johon muut sitä hyödyntävät osat, kuten valvomo, raportointi ja käyttöliittymät liitetään. Lisäksi otetaan kantaa järjestelmän rajoituksiin ja suunnittelussa tehtyihin oletuksiin[1]. Näiden suunnittelutehtävien jälkeen siirrytään yleensä toteutukseen.

Eräs ohjelmistosuunnittelun tärkeimmistä toiminnoista ohjelmiston laadun ja ylläpidettävyyden kannalta on jäljitettävyys. Jäljitettävyydellä osoitetaan eri suunnitteluvaiheiden tuotosten yhteydet toisiinsa. Erityisen oleellisia jäljitettävyystietoja automaatiosovelluksen kannalta ovat vaatimusten jäljitettävyys ne toteuttaviin toimintoihin ja näiden jäljitettävyys edelleen ne toteuttaviin sovelluskomponentteihin. Automaatiosuunnittelussa jäljitettävyyden ylläpito on haastavaa, koska vaatimukset ja toiminnot ja sovel-

[1] Vaikka määrittelyn tulisi olla aukoton, ei kokonaista automaatiojärjestelmää voida määritellä täydellisesti, jolloin joitain seikkoja jää aina suunnittelijan oletuksien ja/tai toimittajan käytäntöjen varaan.

luskomponentit voidaan esittää eri muodoissa, jolloin jäljitettävyys joudutaan toteuttamaan tekstuaalisilla viitteillä tai se jää kokonaan toteuttamatta.

2.4. Nykyisten suunnittelukäytäntöjen kehityskohteita

Automaatio on toimialana varsin konservatiivinen tekniikoiden suhteen. Uusia tekniikoita ja menetelmiä otetaan mielellään käyttöön vasta kun niitä on koeteltu jossain muualla ensin. Osasyynä tähän ovat automaatiojärjestelmien pitkät elinkaaret. Järjestelmissä käytetään jopa vuosikymmeniä vanhoja ratkaisuja, joita näin ollen pidetään toimivina ratkaisuina myös nykyisin. Myös suunnittelumenetelmät ja periaatteet tuntuvat olevan eräänlaista perintömateriaalia, joka kulkee suunnittelijapolvilta toisille. Automaatiosuunnittelusta voidaan tunnistaa useita kehityskohteita [9], joita ovat muun muassa

- yhtenäisen toimialakohtaisen käsitteistön puute
- suunnittelun alustasuuntautuneisuus
- heikot ratkaisuiden uudelleenkäyttömahdollisuudet sekä
- manuaalisen tiedonsiirron tarve suunnitteluosapuolten ja -järjestelmien välillä.

Yhtenäisen toimialakohtaisen käsitteistön puute johtaa helposti sekaannuksiin ja virheellisiin tulkintoihin eri suunnitteluosapuolten tiedonvaihdossa. Tarvitaan yhtenäinen suunnittelukäsitteistö, joka on kaikille osapuolille sama ja kaikkien hyväksymä. Suunnittelun alustasuuntautuneisuus on osittain käsitteistön puutteen johdannaisilmiö. Suunnittelu halutaan saada nopeasti alustariippuvaksi, jotta päästään kiinni edes alustan tarjoamaan käsitteistöön. Tällöin suunnittelussa voidaan säästää, koska pystytään hyödyntämään alustana tarjoamia piirteitä ja ominaisuuksia jo suunnittelun alkuvaiheessa.

Ratkaisuiden uudelleenkäyttömahdollisuudet eri alustoilla heikkenevät alustasuuntautuneisuuden johdosta. Samaa, jo yhdessä projektissa hyväksi havaittua konseptia, ei pystytä sellaisenaan siirtämään toiseen projektiin tai eri alustalle, koska ongelman ratkaisu on määritelty alustariippuvin piirtein varustetuilla käsitteillä. Tässä lähestymistavassa ainoastaan ratkaisun konsepti pystytään hyödyntämään, eikä sitäkään ilman manuaalista työtä, jolla ratkaisu viedään uuteen käyttökohteeseen.

Eri suunnitteluosapuolet käyttävät erilaisia suunnittelujärjestelmiä ja tietomuotoja. Integroituminen eri järjestelmien välillä on heikkoa. Tietoja joudutaan näin siirtämään manuaalisesti eri suunnitteluosapuolten ja -järjestelmien välillä. Tyypillisesti tietoa siirretään käyttäen esimerkiksi Word- tai Excel-dokumentteja, joista suunnittelijat käsin siirtävät tiedot varsinaiseen suunnittelujärjestelmään. Tämä on tehotonta suunnittelijoiden aikaa kuluttavaa työtä, joka ei lisää suunnittelun jalostusarvoa. Lisäksi manuaalinen tiedonsiirto on erittäin altista virheille, joita taas hyvin toteutetussa automaattisessa siirrossa tapahtuu erittäin vähän tai ei ollenkaan.

Jo esitettyjen syiden kannalta automaation (sovellus)suunnitteluprosessia ja menetelmiä kannattaa kehittää. Seuraavissa luvuissa on esitetty periaatteita ja tekniikoita

vaihtoehtoiselle suunnitteluprosessille, joita on hyödynnetty luvussa viisi esiteltävässä automaatiosovellusten mallipohjaisessa kehitysprosessissa.

3. MALLIPOHJAISESTA OHJELMISTOKEHITYKSESTÄ

Edellisen luvun lopussa tarkasteltiin automaatiosuunnittelun nykykäytäntöjen kehityskohteita. Mallipohjainen ohjelmistokehitys tarjoaa mahdollisuuksia automaatiosuunnittelun kehittämiseen varsinkin esitettyjen kehityskohteiden osalta, mutta myös muista näkökulmista tarkasteltuna. Tämän luvun tarkoitus on esitellä lyhyesti AUKOTON-kehitysprosessin taustalla oleva mallipohjainen ohjelmistokehitys sekä tutustuttaa lukija sen periaatteisiin ja tarjoamiin hyötynäkökohtiin ohjelmistokehityksen kannalta.

Mallipohjainen kehitys on (ainakin jossain muodoissa) ohjelmistotuotannossa yleisesti käytetty lähestymistapa ja sen hyviä puolia on haluttu tuoda myös ehdotettuun automaation ohjelmistokehitysprosessin. Luvussa käydään läpi mallipohjaisen ohjelmistokehityksen yleisiä piirteitä ja sen lisäksi esitellään mallipohjaisen ohjelmistokehityksen eräänä ilmentymänä Model Driven Architecture (MDA), jonka pohjalle myös AUKOTON-projektissa kehitetty automaatio-ohjelmistojen mallipohjainen kehitysprosessi on perustettu.

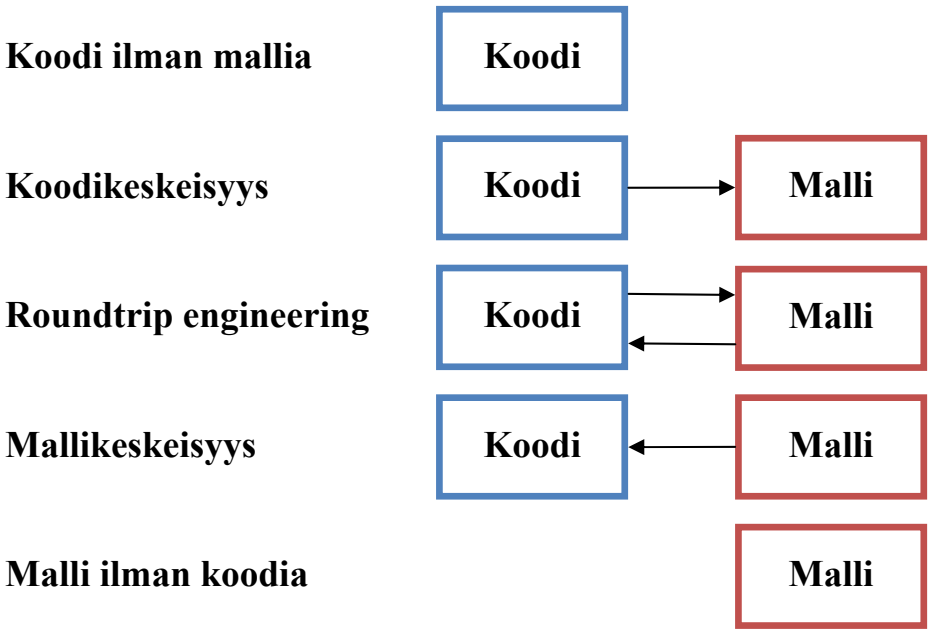
3.1. Yleistä

Mallipohjaiseen kehitykseen, ei siis pelkästään mallipohjaiseen ohjelmistokehitykseen, viitataan yleisesti useilla, lähes samaa tarkoittavilla käsitteillä, joita ovat MDD (Model-Driven Development), MDE (Model-Driven Engineering), MDSD (Model-Driven Software Development) ja MDA (Model-Driven Architecture). Vaikka karkeasti kaikki mainitut mallipohjaisuuteen viittaavat käsitteet tarkoittavatkin lähes samaa asiaa, voidaan käsitteiden välillä nähdä eroja. MDE viittaa enemmän mallipohjaiseen kehitysprosessiin ja suunnitteluun yleensä, eli se ei varsinaisesti ole rajoittunut ohjelmistokehitykseen vaan ottaa kantaa myös prosessiin ja analyysiin arkkitehtuurin ohessa [10 s. 286]. MDSD ja MDA puolestaan viittaavat suoraan ohjelmistokehitykseen. MDSD voidaan nähdä yleisempänä mallipohjaisen ohjelmistokehityksen muotona, joka ei ota kantaa mallinnuskieleen ja on muutenkin yleisempi kuin MDA, joka taas on käytännössä sidoksissa UML-mallinnuskieleen ja muihin OMG:n standardeihin [10 s. 288; 11]. Tämän työn puitteissa keskitytään mallipohjaiseen ohjelmistokehitykseen ja tarkasti ottaen sen variaatioon, eli MDA:han, koska kyse on ohjelmistokehitysprosesseista.

3.1.1. Lähestymistapoja mallien hyödyntämiseen ohjelmistotekniikassa

Ohjelmistokehityksessä voidaan nähdä laaja kirjo tapoja käyttää malleja. Näitä tapoja on esitetty kuvassa 3.1. Ääripäinä esiintyvät koodi ilman mallia ja malli ilman koodia lähestymistavat. Aikoinaan, kun ohjelmat olivat verrattain yksinkertaisia, koodi ilman mallia lähestymistapa oli todennäköisin tapa toteuttaa ohjelmia. Tästä on ajan kuluessa

portaittain siirrytty kohti kuvan 3.1 alalaidassa olevia lähestymistapoja. Tosin nykyäänkin koodi ilman mallia on riittävä toteutusmalli joissakin tapauksissa. On havaittavissa, että koodin merkitys ohjelmointityössä vähenee kun kuvassa 3.1. siirrytään koodi ilman mallia lähestymisestä kohti malli ilman koodia lähestymiseen, jossa koodin tarve lopulta häviää kokonaan.



Kuva 3.1. *Mallien käytön kirjoa ohjelmistotekniikassa (mukailtu lähteistä [12; 13])*

Malleja on siis ohjelmistokehityksessä käytetty jo paljon ennen kuin mallipohjaista ohjelmistokehitystä oli ideoitukaan. Perinteisessä ohjelmistokehityksessä malleilla on ollut kuitenkin rooli lähinnä suunnittelutyön apuvälineenä ja kirjoitetun ohjelmakoodin rakenteen ja toiminnan havainnollistajana sekä dokumentointityökaluna. Tarkoituksena on siis ollut ainoastaan visualisoida kirjoitettua koodia esimerkiksi esittämällä ohjelman luokat ja luokkien väliset suhteet tai etukäteen suunnitella ohjelman rakennetta ja toimintaa. Erilaisia notaatioita ja mallinnuskieliä, kuten UML-kieltä, on sovellettu tähän tarkoitukseen. Tämä toimintamalli edustaa kuvassa 3.1. esitettyä koodikeskeistä mallien käyttöä. Koodikeskeinen lähestymistapa on nykyäänkin yleisessä käytössä oleva tapa toteuttaa ohjelmisto. Mallipohjaisessa ohjelmistokehityksessä tätä ideaa kuitenkin viedään pidemmälle, eivätkä mallit enää vain havainnollista ohjelmaa, eli kirjoitettua ohjelmakoodia, vaan ne määrittävät suoritettavan ohjelman, aivan kuin ohjelmakoodikin. Tähän seikkaan palataan myöhemmin tässä luvussa.

Stahl et al. mukaan mallien käytön perinteisessä lähestymistavassa, eli suunnittelun, visualisoinnin ja dokumentoinnin apuvälineinä, voidaan nähdä muutamia ongelmia. Ensinnäkin ohjelmistoprojekteilla on taipumus olla dynaamisia. Projektin aikana esimerkiksi ohjelman vaatimuksia, rakennetta tai toiminnallisuutta voidaan joutua muuttamaan. Tällöin joudutaan tekemään työtä myös suunnittelu- ja dokumentointimallien

ajan tasalla pitämiseksi. Tämä tehtävä on erityisesti suurten ohjelmistoprojektien osalta osoittautunut monimutkaiseksi ja työlääksi. Toinen havaittu ongelma on se, että mallien käyttö mainittuihin tarkoituksiin ei tuota ohjelmakoodia, vaan ainoastaan edistää sen kirjoittamista, jonka edelleen suorittavat ohjelmoijat. [11]. Vepsäläinen toteaa tämän johtavan siihen, että ohjelmoijat joutuvat tulkitsemaan suunnittelussa käytettäviä malleja ja tuottamaan koodin sen perusteella, miten he malleja ymmärtävät. Vaikka mallien tulisi olla yksikäsitteisiä, eivät ihmiset kuitenkaan pysty tulkitsemaan niitä yksikäsitteisesti. Mahdollisten toteutuksessa tapahtuvien virheiden lisäksi tämä johtaa lopulta virheelliseen lopputulokseen. [14]. Voidaankin kysyä miksi tehtyä mallinnustyötä ei suoraan hyödynnettäisi ohjelman tekemisessä niin, että käsin kirjoitettavan koodin määrä saataisiin vähenemään.

Mallipohjaisessa ohjelmistokehityksessä mallit nähdäänkin hieman toisella tavalla. MDSD:ssä mallit rinnastetaan koodiin, eli malleja käytetään ohjelmiston kuvaamisen pääartefakteina. Tässä lähestymistavassa ohjelman toteutus, eli ohjelmakoodi, seuraa malleista. Periaate on päinvastainen perinteiseen lähestymistapaan nähden, jossa mallit seuraavat ohjelmakoodista. MDSD:n idea onkin nostaa ohjelmointityön abstraktiotasoa. Tarkoituksena on ratkoa ongelmia ongelmakentän, esimerkiksi automaatioalan, käsittein eikä ohjelmointikielen käsittein. Kuvassa 3.1. mallipohjainen ohjelmistokehitys osuu parhaiten kohtiin roundtrip engineering ja mallikeskeisyys, riippuen siitä, kuinka paljon varsinaiseen sovelluskoodiin pitää tehdä käsin muutoksia.

Vastaavaa kehitystä on tapahtunut ohjelmistotekniikan alueella jo pitkään. Siirtyminen binaarisesta ohjelmakoodista assembly-kielellä kirjoitettuihin ohjelmiin ja toisaalta siirtyminen assembly-kielestä kolmannen sukupolven ohjelmointikieliin, kuten C# tai Java, on nostanut ohjelmointityön abstraktiotasoa. Lisäksi muut seikat, kuten käyttöjärjestelmät ja näiden päälle rakennetut ajoympäristöt, kuten JRE (Java Runtime Environment) ovat edelleen nostaneet ohjelmointityön abstraktiotasoa. Suorittimet eivät kuitenkaan ymmärrä assembly-kieltä sellaisenaan, puhumattakaan kolmannen sukupolven ohjelmointikielistä, joten abstraktiotason noustessa on tarvittu transformaatio, joka palauttaa korkean abstraktiotason kuvauksen takaisin suorittimen ymmärtämään muotoon. Transformaation ovat hoitaneet kääntäjät ja linkittäjät, joiden tekeminen on kuitenkin ollut suhteellisen yksinkertaista johtuen tarkasti rajatusta lähtöjoukosta (kielen syntaksi). Mallipohjaisessa ohjelmistokehityksessä lähtöjoukkoa, eli sovellusaluekohtaisia malleja, voidaan ja usein pitääkin laajentaa ja tästä aiheutuu ongelmia transformaattorin määrittelyyn. Aina lähtöjoukkoa ei kuitenkaan ole järkevää pakottaa tiettyyn muotoon. Tällöin ihmisen rooli transformaation suorittamisessa korostuu niin, että ihminen joko suorittaa transformaation tai avustaa siinä.

Perinteisiä ohjelmointitekniikoita ovat leimanneet edellisissä kappaleissa esitetyt ongelmat. Mallipohjainen ohjelmistokehitys pääsee mainituista ongelmista eroon juuri sen takia, että siinä ohjelma kehitetään mallien avulla. Suurimmaksi ongelmaksi nouseekin sellaisen transformaattorin tekeminen, jolla päästään takaisin ajettavaan sovelluskoodiin. Tämä ongelma on kuitenkin kertaluontoinen siihen nähden, että sopiva transformaattori pitää kehittää vain kerran kohdealustalle, jonka jälkeen sitä voidaan

käyttää uudelleen. Sama ilmiö on nähtävissä myös esimerkiksi kolmannen sukupolven ohjelmointikielen osalta siinä, että kääntäjä tehdään kerran kohdealustalle, jonka jälkeen sitä voidaan käyttää mielivaltaiselle järjestelmän mallille, eli tässä tapauksessa ohjelmakoodille.

3.1.2. Hyötynäkökohtia

Käydään läpi muutamia hyötynäkökohtia, joita mallipohjaisella kehityksellä on mahdollista saavuttaa. Stahl et al. listaavat kirjassaan muun muassa seuraavia etuja, joita on mahdollisesti saavutettavissa mallipohjaisen ohjelmistokehityksen avulla [11]:

- lyhyemmät kehitysajat
- korkeampi laatu
- teknologiamuutosten käsiteltävyys
- uudelleenkäyttö sekä
- monimutkaisuuden hallinta abstraktion kautta.

Swithinbank et al. mukaan mallipohjainen ratkaisu (kirjassa käsitellään MDD:tä, mutta samat ominaisuudet sopivat riittävällä tarkkuudella myös MDSD:hen) mahdollistaa lisäksi [15]:

- paremman ylläpidettävyyden
- johdonmukaisuuden sekä
- toimivamman tiedon välittämisen eri osapuolten välillä.

Stahl et al. toteavat, että kehitysaikojen lyheneminen voidaan nähdä seurauksena automaatioasteen noususta. Automaattisten mallitransformaatioiden avulla formaaleista malleista on mahdollista generoida kohdealustalla ajettava sovelluskoodi. [11]. Malleja käytettäessä huomio voidaan sovellusta toteutettaessa kohdentaa suoraan ongelman ratkaisuun samalla abstraktiotasolla sijaitsevilla käsitteillä, jolloin ohjelmointikielen käsitteet voidaan sivuuttaa kokonaan. On tietenkin huomattava, että formaali malli järjestelmästä on hyvin todennäköisesti laajempi ja yksityiskohtaisempi kuin nykyisin käytettävät järjestelmän suunnittelunaikaiset mallit. Näin ollen tällaisen formaalin mallin kehittäminen vie enemmän aikaa kuin tavanomaisen suunnittelunaikaisen mallin. Suunnittelunaikainen malli on kuitenkin järjestelmän formaalin mallin osamalli, joten tässä voidaan nähdä ajansäästömahdollisuus, koska suunnittelunaikaisen mallin lisäksi järjestelmä pitää vielä ohjelmoida. Formaali malli sen sijaan saadaan aikaiseksi suoraan suunnittelunaikaista mallia täydentämällä ja se toimii samalla ohjelmiston dokumentaationa.

Automatisoitu koodingenerointi ja formaalisti määritellyt mallit mahdollistavat laadukkaampien ohjelmistojen tuottamisen [11]. Koska mallipohjaisessa ohjelmistokehityksessä ohjelmakoodi generoidaan suoraan malleista automaattisia transformaatioita käyttäen, poistuvat eri ihmisten tulkintojen aiheuttamat virheet toteutetusta ohjelmakoodista. On tietenkin huomattava, että tämä asettaa vaatimuksia transformaattoreille ja niiden toteuttajille, mutta näin menetellessä mahdolliset ongelmat ovat systemaattisesti

transformaattorissa (olettaen, että sovelluksen mallinnus on virheetöntä). Transformaattori tulkitsee mallin omien sääntöjensä pohjalta aina samoin, eli se tuottaa identtisistä malleista identtisen sovelluskoodin.

Ohjelmistoprojekteissa tulee toisinaan esiin tarve vaihtaa teknologiaa tai halutaan uudelleenkäyttää olemassa olevaa ratkaisua toisella teknologialla. Teknologian vaihdos voi tarkoittaa esimerkiksi toista käyttöjärjestelmää, laitteistoa tai esimerkiksi toisenlaista ajoympäristöä. MDSD tarjoaa helpon tavan teknologiamuutosten hoitamiseen. Koska ohjelman toiminnallinen ja rakenteellinen malli on erotettu teknologian tarjoamista resursseista, jotka on mallinnettu erikseen, on teknologiamuutosten tekeminen mahdollista. Toiminnallinen malli voidaan sellaisenaan siirtää toiselle teknologialle, kunhan teknologia ja siihen sopiva transformaatio on määritelty ja mallinnettu. Käytettäessä olio-ohjelmointia ja sen periaatteita, vastaavia mahdollisuuksia on käytettävissä. Ongelmia saattaa kuitenkin muodostua alun perin esimerkiksi eri ohjelmointikielillä kirjoitettujen toiminnallisuuksien yhdistämisestä. Tätä rajoitetta ei mallipohjaisessa kehityksessä kuitenkaan esiinny samassa mittakaavassa, sillä mallit ovat usein korkeammalla abstraktiotasolla ja transformaatiot eri mallinnusvaiheiden välillä voidaan suorittaa myös manuaalisesti.

Stahl et al. esittävät, että MDSD mahdollistaa ratkaisuiden ja asiatuntijatietämyksen uudelleenhyödyntämisen. Mallipohjaisessa prosessissa jo olemassa olevista arkkitehtuureista, mallinnuskielistä ja transformaatioista muodostuu eräänlainen liukuhihna, jota voidaan hyödyntää erilaisten järjestelmien kehityksessä [11]. Koska MDSD:ssä hyödynnetään alusta- ja teknologiariippumattomia malleja, voidaan alun perin myös eri alustoille suunniteltuja ratkaisuja yhdistää ja näin lisätä uudelleenkäyttömahdollisuuksia. Vastaava ominaisuus on sisäänrakennettu myös ohjelmointikielten käyttöön, koska kääntäjällä tietty koodi voidaan kääntää mille tahansa alustalle ja kääntäjää voidaan tietenkin uudelleenkäyttää.

Ohjelmointityön haaste on tähän asti ollut se, että ongelmia on pitänyt ratkaista ohjelmointikielen käsittein. MDSD lähestymistapa kuitenkin siirtää ongelman ratkonnan ongelmaspesifiin käsitteistöön. Esimerkiksi automaatio-ohjelmistoja rakennetaan automaatioalalle tyypillisillä käsitteillä, kuten säädin ja venttiili, eikä ohjelmointikielikohtaisilla käsitteillä, kuten luokka ja muuttuja. MDSD:ssä abstraktiotaso siis nostetaan ongelmakentän käsitteillä ratkaistavaksi, mikä osaltaan helpottaa monimutkaisuuden hallintaa, koska suuri osa monimutkaisuudesta saadaan mallien avulla abstrahoitua pois näkyvistä ja piilotettua transformaatiossa realisoituvaksi. Abstraktiotason nousua voidaan pitää yhtenä merkittävimmistä mallipohjaisen ohjelmistokehityksen mahdollistamista eduista nykyisiin käytäntöihin nähden. Muut mainitut edut ovat ainakin osin saavutettavissa myös nykyisillä menetelmillä, mutta yksikään niistä ei tarjoa yhtä korkealla abstraktiotasolla olevia ongelmanratkontakäsitteitä.

On syytä korostaa, että mallipohjainen kehitys ei takaa, että yksikään mainituista hyötynäkökohdista realisoituisi mallipohjaista lähestymistapaa käytettäessä. Tärkeämpää on kuitenkin se, että mallipohjainen lähestymistapa tarjoaa mahdollisuuden saavuttaa merkittävää etua nykyisiin menetelmiin verrattuna.

3.2. Model-Driven Architecture

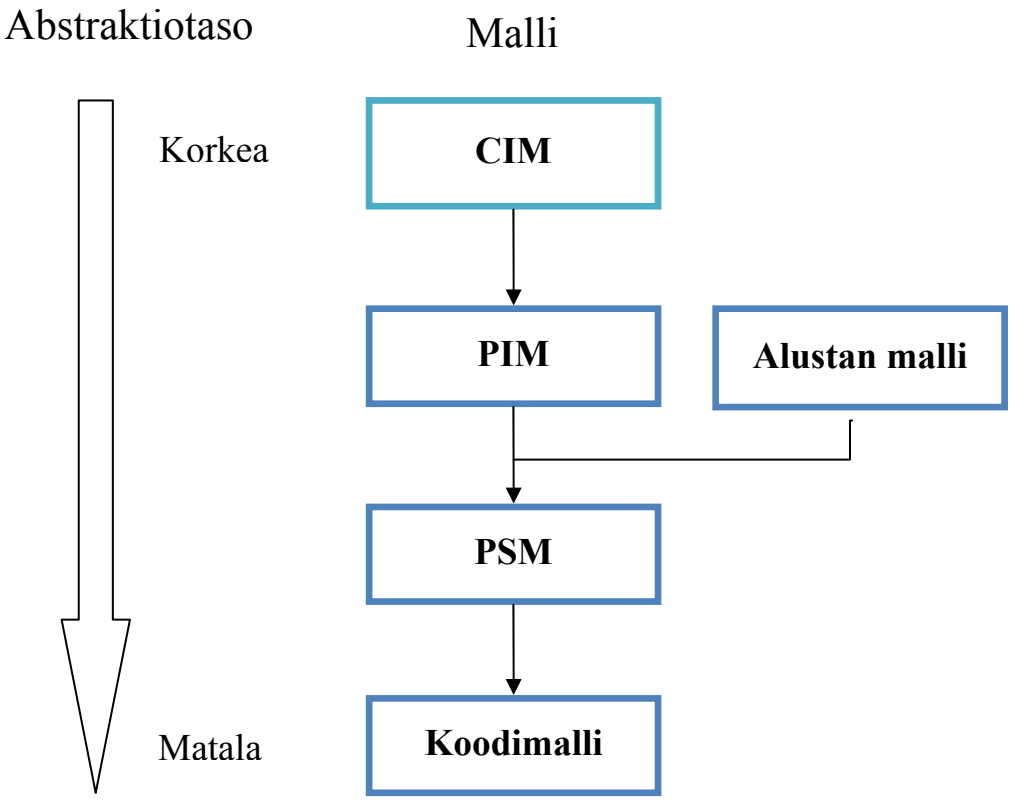
AUKOTON-projektissa kehitetty mallipohjainen kehitysprosessi automaatio-ohjelmistoille edustaa Model-Driven Architecture (MDA) -lähestymistapaa, joka on MDSD:n eräs osajoukko. MDSD ei ole sitoutunut mihinkään mallinnuskieleen, kun taas MDA käyttää mallinnuskielenä UML:ää. Tässä aliluvussa tarkastellaan MDA:n lähestymistapaa hieman tarkemmin.

Model-driven Architecture on Object Management Groupin (OMG) määrittelemä mallikeskeinen ohjelmistokehitysprosessi, jossa järjestelmän toiminnallisuus erotetaan siitä, miten järjestelmä käyttää alustaansa toimintojen suorittamiseen [16]. MDA:ssa (kuten mallipohjaisissa suunnitteluratkaisuissa yleisestikin) mallit ovat pääasiallisia määrittelyartefakteja, eli ohjelmiston toiminta ja rakenne määritellään malleilla, ei ohjelmakoodilla. Eri tasolla olevia malleja luodaan ja linkitetään yhteen ja näin muodostetaan järjestelmän malli [13]. Näin päästään mallipohjaisen ohjelmistokehityksen tavoitteeseen, eli ohjelmointityön abstraktiotason nostamiseen [11]. Tämän ohella MDA:n tavoitteina pidetään siirrettävyyttä, yhteentoimivuutta ja uudelleenkäytettävyyttä. Tähän päästään MDA:n selkeästi standardisoidulla tavalla käyttää UML:ää ja muita ohjelmistostandardeja [16].

3.2.1. MDA-mallien abstraktiotasot

MDA määrittelee neljä eri abstraktiotasoa malleille, joilla järjestelmiä mallinnetaan. Nämä tasot ovat laskentariippumaton malli (CIM, Computation Independent Model) alustariippumaton malli (PIM, Platform Independent Model), alustariippuva malli (PSM, Platform Specific Model) ja alustan malli. Lisäksi yhtenä tasona voidaan pitää koodimallia, vaikka tätä ei OMG:n MDA:ta käsittelevässä ohjeessa [16] mainitakaan, mutta joka kuitenkin seuraa alustariippumattomasta mallista. Eri abstraktiotasojen mallit muodostavat hierarkian, jossa seuraava malli tarkentaa edellistä, eli korkeammalla abstraktiotasolla olevaa mallia [13]. Abstraktiotasojen suhteet on esitetty kuvassa 3.2.

MDA:ssa alustalla tarkoitetaan sellaista joukkoa teknologioita ja alijärjestelmiä, joita alustaa tukeva sovellus voi rajapintojen ja käyttömallien kautta hyödyntää tietämättä alustan tarjoamien palveluiden yksityiskohtaisempaa toteutusta [16]. Alusta on kontekstiriippuva käsite ja se, onko jokin malli alustariippuva vai -riippumaton, riippuu katsantokannasta [12]. Toiselle katsojalle malli saattaa olla alustariippuva ja toiselle alustariippumaton, vaikka kyse olisi täsmälleen samasta mallista. Tätä asiaa käsitellään lisää MDA-prosessin yhteydessä.



Kuva 3.2. *MDA:n mallihierarkia (mukailtu lähteestä [13])*

Laskentariippumaton malli (CIM) on MDA-mallinnusketjussa korkeimman abstraktiotason malli järjestelmästä. OMG:n mukaan se keskittyy kuvaamaan järjestelmän vaatimuksia ja ympäristöä. Rakenteelliset ja toiminnalliset yksityiskohdat ohitetaan tässä mallissa kokonaan, koska ne mallinnetaan vasta matalammilla abstraktiotasoilla. CIM-mallin toteuttajan ei näin ollen tarvitse olla tietoinen niistä malleista tai artefakteista, joilla toiminnallisuus tullaan toteuttamaan. CIM on tärkeä osa mallinnusta, koska se toimii siltana sovelluksen kehittäjien ja toimialakohtaisten asiantuntijoiden välillä. [16]. Laskentariippumatonta mallia voi edustaa esimerkiksi UML:n käyttötapauskaavio [13]. Laskentariippumatonta mallia ei aina käsitellä MDA:n mallitasoja käsittelevissä teoksissa, vaikka se OMG:n lähteissä mainitaankin.

Automaatioalalla asetelma on yleensä edellä kuvatun kaltainen. Automaatiojärjestelmän suunnittelevat usein toimialan asiantuntijat, mutta varsinaisen sovellusohjelmiston taas kehittävät ohjelmistosuunnittelijat. Tässä rajapinnassa osapuolet voisivat kommunikoida juuri laskentariippumattoman mallin välityksellä. On tärkeää, että vaatimukset, joita järjestelmälle on prosessisuunnittelussa asetettu, täytetään sovellusohjelmistolla. Vaatimusten esittäminen ymmärrettävässä muodossa on kuitenkin ollut automaatioalalla jokseenkin vailla yhtenäisiä käytäntöjä. Yhtenäinen vaatimusten mallintaminen yhteisillä käsitteillä voitaisiinkin nähdä ratkaisuna, jossa kaikilla osapuolilla olisi paremmat mahdollisuudet tuottaa ja tulkita tietoa samalla tavalla. Automaatio-ohjelmistojen mallipohjaisessa kehitysprosessissa laskentariippumaton malli realisoituu vaatimuksina, jotka kuvaavat järjestelmän vaatimuksia.

Alustariippumaton malli (PIM) tarkentaa laskentariippumatonta mallia ja on MDA:n malleista toiseksi korkeimmalla abstraktiotasolla. Alustariippumaton malli kuvaa järjestelmää sellaiselta abstraktiotasolta, että mallia on mahdollista käyttää usealla samantyyppisellä alustalla [16]. Tässä annetaan hieman periksi mallipohjaisen ohjelmistokehityksen periaatteelle, jonka mukaan alustariippumaton malli tulisi olla käytettävissä millä tahansa alustalla [11]. Tarkoitus on kuitenkin mallintaa järjestelmän toimintaa sellaisella abstraktiotasolla, että alustan yksityiskohdista eri tarvitse välittää. PIM-vaiheessa mallinnuskielenä voidaan käyttää yleistä tai toimialakohtaista mallinnuskieltä [16]. Yleinen mallinnuskieli voi olla esimerkiksi UML ja toimialakohtainen kieli puolestaan jokin UML-profiili, esimerkiksi kohdassa 4.3 esitelty automaatioprofiili, joka sisältää nimenomaan automaation toimialalle tyypillistä käsitteistöä. UML on saavuttanut lähes standardin aseman mallinnuskielenä ohjelmistotekniikan alueella, joten sitä voidaan käytännössä pitää MDA:n mallinnuskielenä. Tätä tukee myös se seikka, että OMG:n pyrkimyksenä on standardoida menetelmät yhteensopivuuden takia [16].

Automaation sovelluskehityksessä alustariippumaton suunnittelu on vähäistä, kuten suunnittelua käsittelevässä luvussa kaksi todettiin. Osasyynä voidaan pitää sitä, että automatisointiprojektit ovat usein yksilöllisiä, kuten ovat ne prosessitkin, joita automaatiolla ohjataan. Toisaalta suunnittelussa kuitenkin on paljon sellaisia tilanteita, joissa jo kerran suunniteltuja ratkaisuja voitaisiin käyttää uudelleen. Ongelmana on kuitenkin se, että suunnittelu on usein sidoksissa johonkin tiettyyn alustaan, eli jonkin valmistajan tuotteisiin. Tehokkaampaa olisi suunnitella jokin ohjauspiiri, esimerkiksi lämmönsäätimen ohjauspiiri alustariippumattomasti ja sitten sopivalla transformaatiolla muuntaa toiminnan kuvaava malli automaattisella muunnoksella kohdealustalle sopivaksi. Näin kerran suunniteltua ohjauspiirin alustariippumatonta mallia päästäisiin hyödyntämään erilaisten kohdealustojen yhteydessä.

Alustan malli on kuvassa 3.2 esitetty samalla tasolla kuin alustariippumaton malli. Tästä ei kuitenkaan pidä tehdä sitä johtopäätöstä, että kyseessä olisi jonkinlainen alustariippumaton malli. Sen sijaan alustan malli on hyvinkin alustaan sidottu, sillä se mallintaa ne seikat, jotka määrittelevät alustan ja sen tarjoamat palvelut järjestelmälle [16]. OMG:n CORBA (Common Object Request Broker Architecture) on eräs mahdollinen alusta, jolle mallinnettu ohjelma voidaan lopulta asettaa ajoon. CORBA on kommunikaation väliohjelmisto, joka eristää kommunikaation yksityiskohdat sovellukselta [17]. CORBA pystytään mallintamaan riittävällä tarkkuudella, jotta se olisi käytettävissä MDA-mallinnusprosessissa.

Automaatioalalla alustana voi olla esimerkiksi PLC- tai DCS-pohjainen ympäristö, johon on liitetty varsinaiset automaatiolaitteet. Tällaisia alustoja ovat esimerkiksi MetsoDNA, Siemens PCS7, Honeywell Alcont ja ABB 800xA. Alustan mallin tulee kuvata käytettävän alustan tarjoamat palvelut. Ehdotetussa automaatio-ohjelmistojen mallipohjaisessa kehitysprosessissa alustan mallia edustaa UML-profiili, joka sisältää stereotyypit, joilla alustariippumattomat automaatiotoimintojen mallit erikoistetaan tietylle alustalle sopiviksi.

Alustariippuva malli (PSM) kuvaa järjestelmän alustariippuvasta näkökulmasta [16]. Kuten kuva 3.2. esittää, alustariippuva malli rakentuu alustariippumattomasta mallista ja alustan mallista. Riippuen näkökulmasta, alustariippuvaa mallia voidaan joko tarkentaa ja transformoida se uudelleen jollekin toiselle alustalle tai sitten kyse on niin kutsutusta viimeisestä alustariippuvasta mallista. Tällöin malli sisältää kaiken sen informaation, joka on malleilla mallinnettavissa. Tätä mallia voidaan käyttää koodimallin luomiseen.

Koodimalli on matalimman tason malli järjestelmästä. Se on alustariippuvan mallin perusteella generoitu lähdekoodi järjestelmälle. Koodimalli voi olla esimerkiksi JSP-, EJB- tai SQL-koodia [13]. Periaatteessa koodimalli voi olla mitä tahansa koodia, mutta yleensä generoidaan jotain korkean tason ohjelmointikieltä kuten C tai Ada ja varsinainen ajettava koodi käännetään käännöstyökaluilla ajettavaksi sovellukseksi tai ajetaan sopivassa ajoympäristössä, kuten Java-kielet. Automaatiosektorilla koodimalli voi olla esimerkiksi PLC-koodia, jota voidaan asettaa ajettavaksi suoraan PLC-yhteensopiviin laitteisiin.

3.2.2. MDA-prosessi

MDA:n mukainen ohjelmistokehitysprosessi hyödyntää edellisessä aliluvussa esiteltyjä malliabstraktiotasoja. Keskeisenä ideana on mallintaa järjestelmä käyttäen edellä mainittujen abstraktiotasojen malleja. Kukin taso muodostaa oman näkökulmansa järjestelmään ja tarkentaa järjestelmän mallia suhteessa edellisen abstraktiotason malliin. Järjestelmän mallia tarkennetaan jatkuvasti kehityksen edetessä kohti lopullista mallia, josta voidaan generoida järjestelmän sovelluskoodi. Karkeasti MDA-prosessi onkin järjestelmän mallintaminen kullakin abstraktiotasolla ja näiden välissä mallin transformointi seuraavalle abstraktiotasolle. MDA-prosessin kulku on päävaiheittain esitetty kuvassa 3.3.

Kuva 3.3. *MDA-prosessin päävaiheet (mukailtu lähteestä [18 s. 23])*

Vaikka kuvassa 3.3. ei tätä esitetäkään, alkaa MDA-prosessi vaatimusten keräämisellä ja määrittelyllä, kuten mikä tahansa ohjelmistoprojekti. MDA-prosessissa vaatimukset edustavat laskentariippumatonta osaa järjestelmästä, joten ne esitetään laskentariippumattomassa mallissa. Tämä ei ole kuitenkaan välttämätöntä, vaan vaatimukset voidaan esittää myös perinteisin keinoin. Ideaalitilanteessa on kuitenkin olemassa transformaattori, jolla laskentariippumattomasta mallista voidaan generoida alustariippumaton malli. Tämän vaiheen transformaattoreita voidaan pitää vielä melko harvinaisina, koska laskentariippumaton malli sivuutetaan aihetta käsittelevässä kirjallisuudessa lähes kokonaan. Silti, kuten kohdassa 5.3.4 tullaan näkemään, AUKOTON-kehitysprosessissa

laskentariippumaton malli pystytään transformoimaan alustariippumattomaksi malliksi. OMG:n mukaan tavoite on, että laskentariippumattomasta mallista olisi jäljitettävissä ne rakenteet, jotka toteuttavat mallin esittämät vaatimukset järjestelmälle [16]. Myös tämä tavoite toteutuu AUKOTON-kehitysprosessissa hyödynnettävän työkalun myötä.

Laskentariippumattomasta mallista siirrytään alustariippumattomaan malliin joko automaattisen tai manuaalisen mallimuunnoksen avulla riippuen siitä, onko laskentariippumaton malli sopiva transformoitavaksi. Automaattinen mallimuunnos tarkoittaa tässä tapauksessa automaattisen transformaattorin käyttöä, joka muodostaa laskentariippumattomasta mallista pohjan alustariippumattomalle mallille. Manuaalinen mallimuunnos puolestaan olisi alustariippumattoman mallin manuaalinen toteuttaminen laskentariippumattoman mallin pohjalta. Alustariippumattomasta vaiheesta eteenpäin MDA-prosessi etenee kuvan 3.3. mukaisesti. Alustariippumattomassa vaiheessa mallia tarkennetaan kohti tilannetta, jossa se kuvaa järjestelmän toiminnallisuuden ottamatta kantaa alustaan. Kun alustariippumaton malli on saatu riittävän tarkaksi, se transformoidaan työkalun avulla tai manuaalisesti alustariippuvaksi malliksi tai malleiksi. Tämä onkin yleensä koko MDA-prosessin monimutkaisin vaihe [18]. Alustariippuvaa mallia tarkennetaan edelleen kohti tilaa, jossa se mahdollisimman tarkasti mallintaa halutun järjestelmän ottaen huomioon alustariippuvat seikat. PSM-malli on yleensä tiukasti sidoksissa alustaan, joten sen transformaatio koodimalliksi on yleensä suoraviivainen prosessi [18].

Jos kuvan 3.3. suoraviivaista prosessia noudettaisiin tarkalleen, kuten edellä on kuvattu, voisi PIM- ja PSM-mallien väliin jäädä abstraktiotasojen välinen kuilu, jota olisi hankala ylittää transformaatiolla [13]. OMG:n mukaan prosessia voidaan kuitenkin jakaa useampaankin osaan kuin mitä kuvassa 3.3 on esitetty, eli malleja voidaan muodostaa useammalle abstraktiotasolle [16]. Tällöin abstraktiotasojen välinen kuilu kapenee ja transformaatio on helpommin suoritettavissa. Jos abstraktiotasojen lukumäärää nostetaan, ajaudutaan helposti tilanteeseen, jossa PIM- ja PSM-mallien erot hämärtyvät, koska Brownin mukaan yksi malli voi olla yhden vaiheen PIM ja toisen PSM [12]. On kuitenkin huomattava, että vaikka alustariippumaton suunnittelu jaettaisiinkin pienempiin osiin, on se silti alustariippumatonta suunnittelua.

Mallien transformaatioista voidaan todeta sen verran, että OMG esittelee ohjeistuksessaan [16] lukuisia eri tapoja suorittaa mallitransformaatioita. Transformaatioiden toimintatapa ja mekanismit eivät kuitenkaan ole tämän diplomityön kannalta oleellisia, joten niitä ei käsitellä.

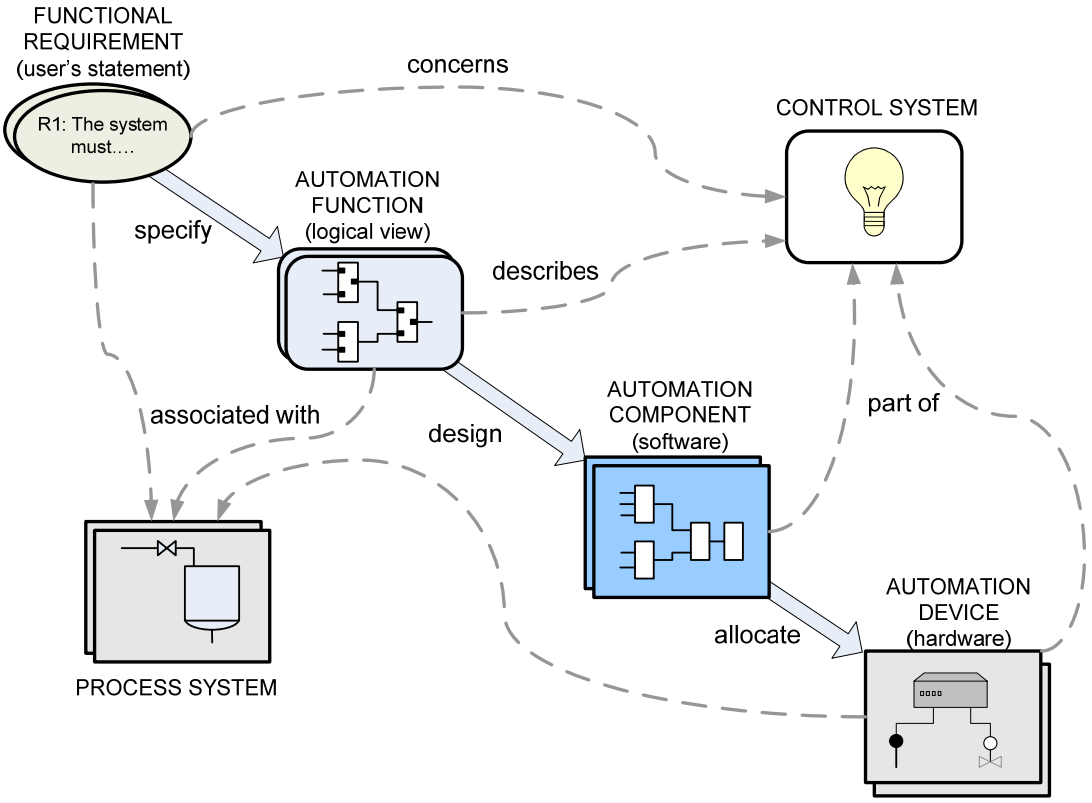
Vaikka MDA:lla on paljon hyviä ominaisuuksia, ei MDA-konsepti vielä nuoresta iästään johtuen ole täysin valmis ratkaisu. Laajamittaista tukea tarjoavien työkalujen vähyys ja teknologian kypsymättömyys vaikeuttavat toistaiseksi MDA-prosessiin siirtymistä laajassa mittakaavassa. Voidaan kuitenkin nähdä, että hiljalleen MDA-prosessi tulee valtaamaan alaa sen tarjoamien hyvien ominaisuuksien ansiosta. AUKOTON-projektin kaltaiset kehityshankkeet, joissa mallipohjaista lähestymistapaa sovelletaan, tuetaan työkaluketjulla ja kehitetään räätälöityjä ratkaisuja kohdennetulle alueelle, vievät mallipohjaista kehitystä eteenpäin.

4. MALLIPOHJAISTEN AUTOMAATIO-OHJELMISTOJEN KEHITYSYMPÄRISTÖ

Kuten luvussa kaksi tuli esille, automaatioalalla on perinteisesti käytetty eriäviä käsitteitä ja epäyhteensopivia suunnittelutyökaluja eri suunnitteluosapuolten välillä. Uuden mallipohjaisen lähestymistavan ehtona on kuitenkin yhtenevä käsitteistö ja sen saumaton siirtyminen eri suunnittelutoimijoiden ja -työkalujen välillä. Tässä luvussa esitellään automaatio-ohjelmistojen kehitysympäristö, joka on kehitetty mallipohjaista lähestymistapaa varten. Kehitysympäristö koostuu yleisistä kehitysprosessin käsitteistä, niiden UML-implementaatiosta eli UML-automaatioprofiilista ja UML AP-työkalusta, joka tukee automaatioprofiilin käyttöä. Kehitysympäristö tarjoaa työkalu- ja käsiteresurssit, joilla mallipohjaisen kehityksen edut saadaan automaatioalalla käyttöön. Luku etenee muuten yllä mainitussa järjestyksessä, mutta ennen UML-automaatioprofiilin läpikäyntiä tarkastellaan UML-kieltä lyhyesti yleisemmällä tasolla ja tutustutaan sen laajennettavuusmekanismiin, jota hyödyntämällä myös automaatioprofiili on toteutettu.

4.1. Yleinen käsitteistö

AUKOTON-kehitysprosessia tuetaan käsitteistöllä, joka määrittelee eri suunnitteluvaiheiden roolin kehitysprosessissa. Konseptit ja niiden linkittyminen säätö- ja prosessijärjestelmään on esitetty kuvassa 4.1. Kuten kuvasta voidaan nähdä, käsitteet ovat neljällä tasolla, jotka ovat vaatimus, automaatiotoiminto, automaatiokomponentti ja automaatiolaite. Käsitteet muodostavat konkretisoituvan ketjun, jossa seuraava käsite on aina edellistä lähempänä lopullista toteutusta. Käsitteet ja konseptit käydään tarkemmin läpi seuraavissa aliluissa.



Kuva 4.1. *AUKOTON-projektin konseptien yleiskatsaus [9]*

4.1.1. Vaatimus

Ensimmäinen käsite AUKOTON-projektin käsitteistössä on vaatimus. Vaatimus määrittelee mitä järjestelmän tulee tehdä ja millaisten reunaehtojen puitteissa. Vaatimukset siis vastaavat kysymykseen, mitä järjestelmän tulisi tehdä ja millä ehdoilla, eli mitkä ovat toiminnan rajoitteet. Vaatimus voisi esimerkiksi määritellä, että järjestelmässä olevan moottorin pyörintänopeutta pitää voida säätää (mitä) ja pyörintänopeuden tulee pysyä +-10 1/min etäisyydellä asetetusta pyörintänopeudesta kuormituksen muutosten yhteydessä (millä reunaehdoilla).

Vaatimukset ottavat kantaa (tai ainakin niiden tulisi ottaa kantaa) automaatiojärjestelmään täysin toteutusriippumattomasti. Toteutusriippumattomuudella tarkoitetaan vaatimusten yhteydessä sitä, että niillä ei oteta kantaa siihen, millä tekniikoilla ja menetelmillä vaatimukset saadaan toteutettua. Vaatimukset saattavat ottaa kantaa joihinkin toteutuksen konsepteihin, mutta konsepteja ei kuitenkaan tulisi vielä tässä vaiheessa pyrkiä määrittelemään liikaa tai liian tarkasti. Vaatimukset määrittävät automaatiotoimintoja, joita tarkastellaan seuraavassa aliluvussa.

4.1.2. Automaatiotoiminto

Automaatiotoiminto edustaa järjestelmän ”toiminnallisuuden yksikköä”, eli itsenäistä tai koosteista entiteettiä, joka määrittelee jonkin automaatiojärjestelmän yksittäisen toiminnallisuuden, eli toteuttaa yhden tai useamman järjestelmältä vaaditun toiminnon [19].

Tällainen voi olla esimerkiksi ohjauslähtö, säätöpiiri tai -sekvenssi. Toiminnot voivat koostua muista automaatiotoiminnoista tai niitä voidaan hyödyntää itsenäisinä.

Automaatiotoiminnot kuvaavat järjestelmän automaatiokonseptin, eli sen, minkälaisilla automaation ”rakennuspalikoilla” vaatimuksissa määritelty automaation toiminnallisuus saadaan toteutettua. Siinä missä vaatimukset ottavat kantaa järjestelmältä vaadittuun toimintaan ja toiminnan reunaehtoihin, automaatiotoiminnot kuvaavat automaatiojärjestelmän toimintoja prosessisuunnittelijan ja prosessin toiminnallisuuden näkökulmasta [19]. Automaatiotoiminnoilla voidaan määritellä esimerkiksi säätöpiiri, joka mittaa moottorin pyörintänopeutta, laskee mittauksen ja asetusarvon perusteella ohjauksen ja lähettää ohjauksen moottorille. Tässä yksittäisiä automaatiotoimintoja ovat esimerkiksi säätöpiiri, säätöalgoritmi, mittaussisääntulo ja ohjauslähtö.

Automaatiotoiminnot kuvaavat järjestelmää edelleen alustariippumattomasti. Automaatiotoiminnot eivät siis ota kantaa järjestelmän toteutusriippuviin seikkoihin, kuten tarkkaan parametrointiin tai alustan palveluiden käyttöön. Ne kuvaavat ohjaus- tai säätöjärjestelmää ja ovat näin osa prosessisysteemiä.

4.1.3. Automaatiokomponentti ja -laite

Tommilan & Mätäsniemen mukaan automaatiokomponentti edustaa ohjelmistomoduulia säätöjärjestelmän implementaatiossa. Automaatiosovellukset koostuvat yhdestä tai useammasta automaatiokomponentista. [19]. Automaatiokomponentit ottavat kantaa siihen, millä eri tavoilla automaatiotoiminnot voidaan toteuttaa suoritettaviksi ohjelmistokomponenteiksi. Automaatiotoimintohan ei pidä sisällään ohjelmistoarkkitehtuuriin tai toteutusalustaan liittyviä yksityiskohtia, joita tarvitaan ajettavan sovelluksen tuottamiseen. Automaatiotoiminnot voidaankin nähdä automaatiokomponenttien abstraktioina. Automaatiokomponentti sisältää kaiken sen informaation, jota ajonaikainen sovelluskomponentti tarvitsee (mukaan lukien esimerkiksi parametrointi). Automaatiokomponentit ovat näin ollen osa lopullista ohjaus- tai säätöjärjestelmää.

Toisen osan säätöjärjestelmästä muodostaa automaatiolaitteisto. Se edustaa fyysistä automaatiojärjestelmän laitteistoa kuten säätimiä, toimilaitteita ja antureita sekä tietokonelaitteistoa liityntöineen. Automaatiolaitteisto suorittaa automaatiokomponenteista muodostettua automaatiosovellusta, eli automaatiokomponentit allokoidaan sopiville automaatiolaitteille ajettaviksi. Automaatiolaitteisto voidaankin sovelluksen kannalta nähdä resurssina sovelluksen suorittamiselle. [19]

4.2. UML

UML (Unified Modeling Language) on Rational Software Corporationin luoma ja OMG:n (Object management Group) myöhemmin standardoima graafinen mallinnuskieli. Hieman tarkentaen UML on standardi kieli ohjelmistojen määrittelyyn, visualisointiin, laatimiseen ja dokumentointiin. Se edustaa kokoelmaa suunnittelukäytäntö-

jä[2], jotka ovat osoittautuneet toimiviksi laajojen ja monimutkaisten järjestelmien mallintamisessa [20]. UML-kielen alkuperäinen versio 0.9 julkaistiin vuonna 1996 [20], jonka jälkeen sitä on kehitetty OMG:n toimesta ja se on kirjoitushetkellä versiossa 2.2. UML-kielen uudet versiot sisältävät laajennuksia ja parannuksia vanhempiin versioihin nähden, mukaan lukien parempi tuki mallipohjaiselle ohjelmistokehitysprosessille [11].

UML:ää voidaan Fowlerin mukaan hyödyntää kolmella tasolla: luonnostelussa, määrittelyssä ja suunnittelussa sekä ohjelmointikielenä. Luonnostelussa UML:ää käytetään kommunikointivälineenä suunnittelijoiden kesken ja sillä esitetään karkeita kuvauksia järjestelmästä. Kuvauksia voidaan tehdä valmiista koodista, jolloin kyse on dokumentoinnista tai kuvauksia tehdään koodille, joita käytetään ohjelmoinnin pohjana. Määrittelyssä ja suunnittelussa UML-kielellä kuvataan järjestelmä yksityiskohtaisemmin. Kuvauksen tarkkuustaso pyritään saattamaan sellaiseksi, että koodin kirjoittajan ei enää tarvitse tehdä merkittäviä suunnittelupäätöksiä.

Käyttö ohjelmointikielenä on kuitenkin tämän diplomityön kannalta mielenkiintoisin UML:n hyödyntämismuoto. Tässä lähestymistavassa UML-malleja käytetään ohjelmointikielen sijaan ajettavan sovelluskoodin generoinnissa. Toisin sanoen UML-malli ”käännetään” suoraan ajettavaksi koodiksi, eikä varsinaista lähdekoodia tarvitse kirjoittaa. Tätä lähestymistapaa kutsutaan yleisesti mallipohjaiseksi kehitykseksi, jota käsiteltiin aiemmin tässä diplomityössä. [21]

Perusmuodossaan UML määrittelee joukon mallinnuselementtejä ja kaaviotyyppejä, jotka keskittyvät (olio-)ohjelmistojen kuvaamiseen. Järjestelmä kuvataan mallinnuselementeillä ja niiden suhteilla käyttäen tarkoitukseen sopivia kaaviotyyppejä, kuten luokka-, tila- ja käyttötapauskaavioita. Kaaviotyyppejä on erilaisia (yhteensä kolmetoista kappaletta), joista kukin sopii erilaisiin tarkoituksiin. OMG:n mukaan kaaviotyypit jakautuvat kolmeen kategoriaan, joista ensimmäiseen kuuluvat tyypit määrittelevät järjestelmän rakennetta, esimerkiksi luokka- ja komponenttikaaviot. Toiseen kategoriaan kuuluvat toiminnallisuutta määrittelevät kaaviotyypit, joita ovat esimerkiksi tila- ja aktiviteettikaaviot. Kolmannen kaaviotyyppiluokan muodostavat interaktiokaaviot, joihin kuuluvat muun muassa sekvenssi- ja ajoituskaaviot [22]. Vaikka UML on ohjelmistotekniikkaan suuntautunut, sen käyttö ei kuitenkaan rajoitu ainoastaan tälle alueelle. UML-kieltä onkin hyödynnetty muun muassa terveydenhuollossa, sotilaallisissa tarkoituksissa, hajautetussa laskennassa ja vähittäismyynnissä [23]. Profiilien avulla UML voidaan erikoistaa lähes mille tahansa toimi- tai kohdealueelle sopivaksi [24].

4.2.1. Metamallinnuksesta

Tässä aliluvussa käsitellään lyhyesti metamallinnusta, jotta voidaan paremmin ymmärtää UML-profiilimekanismin toimintaa, joka esitellään seuraavassa aliluvussa. Metamallit ovat tässä oleellisia siinä suhteessa, että niiden avulla voidaan muun muassa mää-

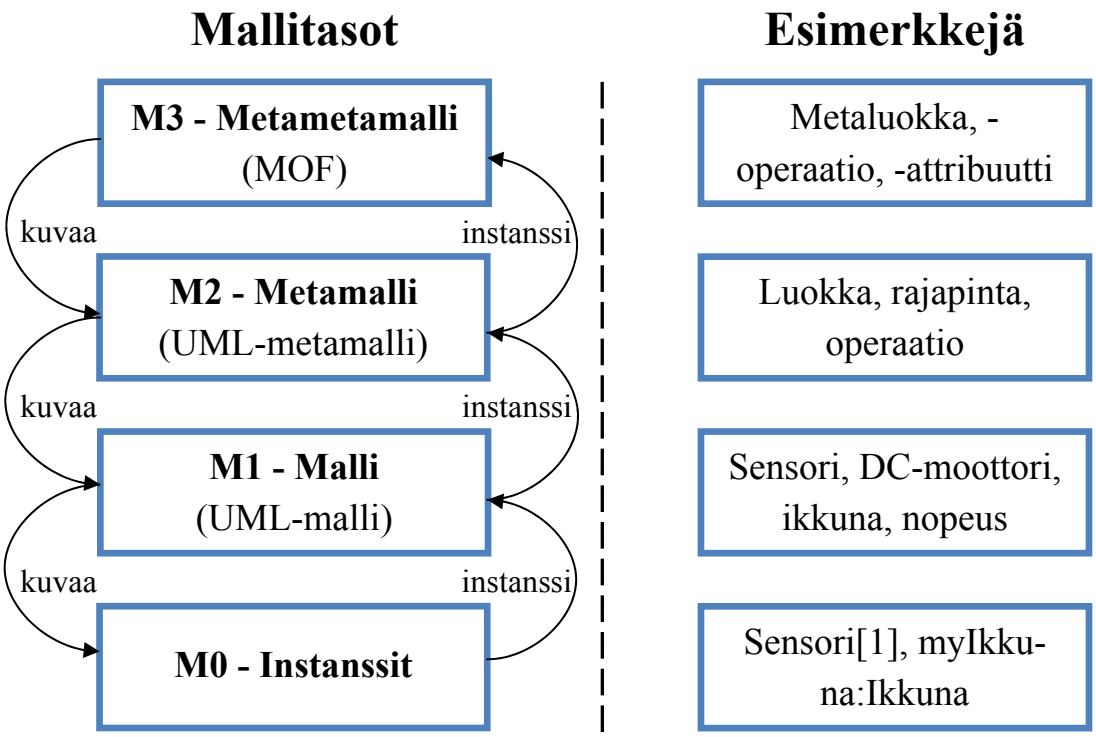
[2] Ennen UML:n ilmaantumista ohjelmistotekniikan alueella oli suuri joukko kilpailevia kuvausmetodeja, joista yhdelläkään ei kuitenkaan ollut selkeästi määräävää asemaa. UML, jota nykyisin voidaan pitää määräävänä kuvausmetodina, on osaksi rakentunut näiden kuvausmetodien pohjalta. [20]

ritellä toimialakohtaisia mallinnuskieliä. Ne ovat oleellisessa asemassa myös muissa mallipohjaisen (ohjelmisto)kehityksen ongelmakohtien ratkaisuissa [11].

Metamallit ovat malleja, jotka määrittelevät mallintamista. Mallinnuskielen metamalli kuvaa mallinnuskielen abstraktin syntaksin ja staattisen semantiikan. Käytännössä metamalli siis määrittää mallinnuskielen käsitteet, niiden suhteet ja mahdollisesti rajoituksia ja sääntöjä koskien mallintamista. [11; 14]

Metamallin ja mallin välillä vallitsee luokka-instanssi-suhde, eli jokainen malli on metamallinsa instanssi. Metamalli siis kuvaa mallia ja malli on metamallin instanssi. Periaatteessa myös metamallilla tulisi siis olla oma metamalli, eli metametamalli, joka määrittelee metamallin. Näin ollen metamallilla A tulisi olla metametamalli B, joka määrittelee metamallin A. Ketjua voitaisiin jatkaa loputtomiin, mutta yleensä ongelma on ratkaistu määrittelemällä metamalli, joka on itsekantava. Itsekantava metamalli määrittelee itse itsensä tai se on määritelty jotenkin muuten. Metamallihierarkia, johon UML kuuluu, on nelitasoinen. [11]

Tarkastellaan seuraavaksi OMG:n nelitasoista metamallihierarkiaa, joka on esitetty kuvassa 4.2. Esityksessä vasemmalla on kuvattu metamallitasot ja oikealla annettu esimerkkejä siitä, mitä kyseinen taso käytännössä kuvaa. Tason M0 esimerkit pyrkivät kuvaamaan ajonaikaisia instansseja, vaikka ne on esitetty ohjelmointikielen syntaksilla.



Kuva 4.2. *UML-kielen metamallitasot ja esimerkkejä niiden kuvaamista asioista (mukailtu lähteistä [14; 21])*

UML ei ole hierarkian ylimmällä mallitasolla, joten se voidaan nähdä sekä mallina että metamallina. UML:n sijaan hierarkian ylimmällä mallitasolla (M3) on MOF (Meta-Object Facility), joka on tässä metamallihierarkiassa metametamalli. Se on siis kieli, jolla voidaan kuvata mallinnuskieliä, eli metamalleja (esimerkiksi UML-metamalli). Mallitasoa M3 tarvitaan mallitason M2 transformaatioiden määrittelyyn. Koska MOF:llä ei ole metamallia, se on määrittelyn suhteen itsekantava, eli MOF määrittelee itse itsensä. Hierarkiassa toiseksi ylimmällä mallitasolla (M2) oleva UML-metamalli määrittelee mallinnuksessa käytettävän käsitteistön, johon kuuluvat muun muassa luokka ja assosiaatio. UML-metamalli on toisaalta MOF:n instanssi, eli malli. Toisaalta se on, kuten nimestä voi päätellä, metamalli seuraavalla mallitasolla (M1) olevalle UML-mallille. UML-malli sisältää ne UML-metamallin instanssit, kuten esimerkiksi luokat ja niiden väliset assosiaatiot, joiden avulla suunnittelija laatii UML-kaavioita. UML-mallissa luokka voi mallintaa esimerkiksi sensoria sekä säädintä ja assosiaatio taas niiden välistä yhteyttä. Hierarkian alin mallitaso (M0), kuvaa reaalimaailman oliota. Nämä instanssit edustavat reaalimaailman ilmentymiä, joita UML-malleilla määritetään. Kun hierarkiassa edetään mallitasolta M3 mallitasolle M0, elementtien määrä mallitasoa kohti kasvaa. Tämä on luonnollista, koska esimerkiksi mallitason M2 metaelementillä ”luokka” on yleensä useita instansseja UML-mallissa (M1), esimerkiksi auto, reitti ja kartta.

4.2.2. UML-profiilimekanismi

UML on perusmuodossaan ohjelmistotekniikkaan painottunut, eli sen mallinnuskäsitteistö ja kaaviotyypit on suunniteltu ohjelmistojen, varsinkin olio-ohjelmistojen, mallinnusta silmälläpitäen. UML:än on kuitenkin rakennettu mekanismi, jolla sitä voidaan laajentaa ja erikoistaa myös muille toimialoille ja käyttökohteille sopivaksi. Tämä mekanismi on UML-profiili, joka on käytännössä UML jollain (vähäisellä) tavalla erikoistettuna [24]. UML-profiilit ovat ihmisen ja työkaluohjelmiston ymmärtämiä ohjeistoja, joilla UML-kielen käsitteistöä voidaan tarkentaa ja rajoittaa jollekin tietylle toimialalle tai sovellusalueelle sopivaksi. Tarkennukset ja rajoitteet mallinnetaan stereotyyppien, ominaisuus-arvoparien ja rajoitteiden avulla. [25]

Edellisessä aliluvussa mainittiin, että UML-mallinnus suoritetaan mallitasolla M1 metamallitason M2 käsitteitä hyödyntäen. UML-profiilit laajentavat UML-metamallia eli M2-tason metamallia. Metamalliin lisätään stereotyyppejä, ominaisuus-arvopareja ja rajoitteita [25]. Stereotyypit liittyvät aina johonkin metamallissa jo olemassa olevaan elementtiin. Esimerkiksi UML-metamallissa olevaan luokka-elementtiin voidaan liittää stereotyyppi PID-säädin, jolla luokka voidaan erikoistaa PID-säädintä mallintavaksi. Metamalliin voidaan luoda myös täysin uusia metamallielementtejä [25]. UML-metamalliin voidaan siis lisätä esimerkiksi uusi PID-säädin elementti, joka ei kuulu UML-metamallin peruselementteihin. Metamalliin lisättävien elementtien tulee kuitenkin olla metametamallin (MOF) mukaisia.

UML-profiileilla voidaan ohjeistaa ja standardoida UML:n käyttöä tietyllä sovellus- tai toimialalla. Näin pystytään tarjoamaan alalle oma käsitteistö. Tämä on erityisen tär-

keää sellaisilla aloilla, joilta yhtenäinen käsitteistö puuttuu. Profiilit mahdollistavat UML-pohjaisen sovellusaluekohtaisen mallinnuskielen, joka kuitenkin on UML:n piirissä ja näin ollen olemassa olevaa työkalutukea pystytään hyödyntämään. Profiileilla on kuitenkin mahdollista laatia perus UML-mallien kanssa (osittain) epäyhteensopivia malleja. Tällöin vaaditaan tukea myös mallinnuksessa käytettäviltä työkaluilta. Tuki voi kuitenkin käytännössä olla vähäistä varsinkin, jos profiili ei ole saanut merkittävää asemaa. Olemassaolevia ja kohtuullisen hyvin tuettuja UML-profiileja ovat muun muassa UML RT-profiili (UML-profile for Schedulability, Performance and Time specification) [26] ja SysML (Systems Modeling Language) [27].

4.3. UML-automaatioprofiili

Vaikka UML-kielellä voidaan periaatteessa kuvata lähes minkä tahansa ohjelmiston rakennetta ja toimintaa, ei se sellaisenaan sovi automaatiosovellusten kuvaamiseen kovin hyvin. Tämä johtuu siitä, että UML-kieltä ei alun perin suunniteltu käytettäväksi automaatio-ohjelmistojen, vaan yleisten olio-ohjelmistojen kuvaamiseen, eikä se näin ollen sisällä automaatiosovellusten kuvaamisessa tarvittavaa erikoiskäsitteistöä.

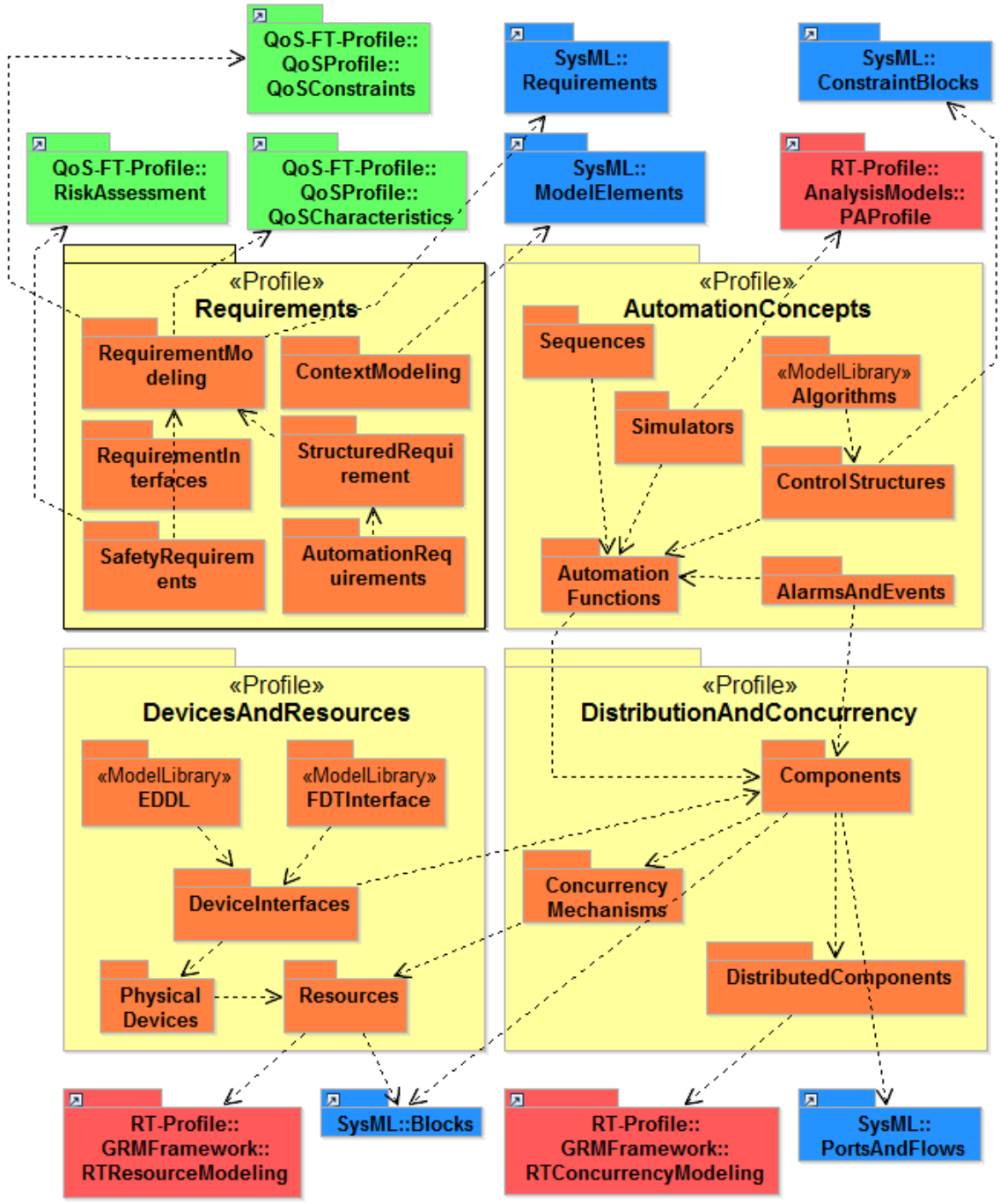
UML-automaatioprofiili on alun perin USVA-hankkeessa toteutettu UML-kielen laajennus, eli kohdassa 4.2.2 esitetyn profiilimekanismin mukainen UML-profiili. Automaatioprofiili määrittelee automaatioalalle tyypillisiä käsitteitä, joilla automaatiosovelluksia voidaan mielekkäästi kuvata. Tällaisia käsitteitä ovat muun muassa säätösilmukka, säädin ja toimilaite, joille jokaiselle on profiilissa esitetty määrittely. Profiilin tarkoitus on määritellä automaatiosovellusten kehitysprosessissa ja yleisemmin koko automaatioalalla hyödynnettävissä oleva toimialakohtainen käsitteistö. Automaatioprofiililla onkin potentiaalia toimia automaatioalaa yhtenäistävänä standardina, koska se määrittelee alalla käytetyt käsitteet yksikäsitteisesti ja kootusti siten, että kaikilla toimijoilla on mahdollisuus käyttää samaa käsitteistöä.

Ritalan mukaan automaatioprofiilin suunnitteluperusteina ovat olleet mallinnuskonseptien tuttuus, uudelleenkäyttö sekä laajennukset, modulaarisuus ja laajennettavuus. Profiilin mallinnuselementtien tulee olla käsitteellisesti tuttuja automaatiosuunnittelijoille, jotta he pystyvät mallintamaan automaation sovellusohjelmistoja niiden avulla. Profiilin mallinnuselementit on pääasiallisesti uudelleenkäytettyjä ja laajennettuja versioita jo olemassa olevista UML:n ja sen profiilien mallinnuselementeistä. Jo olemassa olevista UML-profiileista automaatioprofiili hyödyntää muun muassa RT- ja SysML-profiileja (katso kuva 4.3). Modulaarisuus on saavutettu jakamalla mallinnuselementit pienehköihin pakkauksiin, joilla on tiukasti rajattu vastuualueensa. Profiili tarjoaa laajennettavuutta mahdollistamalla käyttäjätason laajennuksia osaan mallinnuselementeistä siten, että niitä voidaan tarkentaa paremmin tarkoitukseensa sopiviksi. [28]. Näillä periaatteilla profiilista on pyritty saamaan mahdollisimman hyvin tehtäväänsä sopiva UML-kielen laajennus, jonka käsitteet ovat toisaalta riittävän lähellä nykyisiä suunnittelukonsepteja ja toisaalta ilmaisuvoimainen ja laajennettava profiili, jolla on edellytyksiä vastata myös tulevaisuuden haasteisiin.

Automaatioprofiili on tällä hetkellä kehityksen alla. Tällä hetkellä erityisesti profiilin vaatimus- ja automaatiokonseptialiprofiilit ovat kokemassa muutoksia. Profiilia pyritään kehittämään ja laajentamaan sellaiseksi, että se mahdollisimman hyvin vastaisi suunnittelijoiden tarpeita ja pystyisi palvelemaan koko toimialaa tarjoamalla yhtenäisen käsitteistön. Tässä diplomityössä käsitellään automaatioprofiilin versioita 0.14, joka on täydennetty profiiliin liitettävällä päivityksellä, jota ei toistaiseksi ole implementoitu profiilin viralliseen versioon.

4.3.1. Automaatioprofiilin arkkitehtuurista

Automaatioprofiilin arkkitehtuuri on esitetty kuvassa 4.3. Profiili koostuu neljästä aliprofiilista, jotka ovat Requirements, AutomationConcepts, DistributionAndConcurrency ja DevicesAndResources. Profiili hyödyntää UML:n peruselementtien lisäksi SysML [27], RT- [26], QoS-FT-profiileja [29] mallielementtien pohjana.



Kuva 4.3. *Automaatioprofiilin yleiskatsaus. Profiili koostuu neljästä aliprofiilista, joiden pakkauksilla on riippuvuuksia toisiinsa ja muihin UML-profiileihin.*

Kullakin aliprofiililla on oma vastuualueensa profiilissa, eli kukin aliprofiili sisältää tiettyyn sovelluksen kehitysvaiheeseen ja alueeseen liittyviä käsitteitä. Aliprofiileja ei kuitenkaan voida pitää täysin itsekantavina vaan niiden elementeillä on riippuvuuksia toisiinsa sekä muihin UML-profiileihin [14]. Aliprofiilien vastuualueet ovat lyhyesti seuraavanlaiset: Requirements sisältää vaatimusten mallintamiseen liittyviä käsitteitä, AutomationConcepts sisältää alustariippumattomaan järjestelmän toiminnalliseen mallintamiseen liittyviä käsitteitä, DistributionAndConcurrency sisältää hajauttamiseen, rinnakkaisuuteen ja sovelluksen arkkitehtuuriin liittyviä käsitteitä, DevicesAndRe-

sources-aliprofiili puolestaan sisältää erilaisten resurssien, fyysisen laitteiston ja laitteistorajapintojen kuvaamiseen liittyvää käsitteistöä. Aliprofiilit koostuvat edelleen pakkauksista, joilla on omat selkeät vastuualueensa aliprofiilin sisällä, samaan tapaan kuin aliprofiileilla on profiilissa. Pakkaukset esitellään tarkemmin varsinaisten aliprofiiliesittelyiden yhteydessä. [28]

Kappaleissa 4.3.3-4.3.6 esitellään tarkemmin automaatioprofiilin aliprofiilit. Aliprofiilit käydään läpi pakkaustasolla esitellen niiden käyttötarkoitusta esimerkkien avulla. Tarkemmat kuvaukset kustakin automaatioprofiilin pakkauksesta ja elementistä löytyvät varsinaisesta ohjeistusdokumentista [30]. Ennen aliprofiileja tarkastellaan lyhyesti automaatioprofiilin määrittelemiä kaaviotyyppejä.

4.3.2. Kaaviotyypit

UML-automaatioprofiili mahdollistaa UML:n ja SysML:n kaaviotyyppien hyödyntämisen järjestelmän mallin rakentamisessa ja kuvaamisessa. Lisäksi profiili määrittelee kolme uutta kaaviotyyppiä, jotka ovat laajennuksia UML:n ja SysML:n kaaviotyypeistä. Uudet kaaviotyypit ovat vaatimusmäärittelykaavio (RequirementSpecificationDiagram), säätörakennekaavio (ControlStructureDiagram) ja automaatiosekvenssikaavio (AutomationSequenceDiagram). Kullakin kaaviotyypillä on oma määrätty käyttötarkoitus ja syntaksi. Kaavioon voidaan liittää vain sellaisia elementtejä, jotka ovat kaavion kannalta merkityksellisiä ja jotka sopivat kaavion käyttötarkoitukseen. [28]

Vaatimusmäärittelykaaviota käytetään järjestelmän vaatimusten esittämiseen. Kaaviotyyppi on laajennettu SysML:n vaatimuskaaviosta (RequirementDiagram) ja sitä käytetään Requirements-aliprofiilin käsitteiden kanssa. Esimerkkejä vaatimusmäärittelykaaviosta voi löytää Requirements-aliprofiilin esittelystä (aliluku 4.3.3) esimerkiksi kuvista 4.5-4.7.

Säätörakennekaaviolla kuvataan automaatiojärjestelmän säätötoiminnallisuutta ja -rakenteita. Tämä on profiilin määrittelemistä kaaviotyypeistä monipuolisin ainakin niiden käsitteiden osalta, joita voidaan sen yhteydessä hyödyntää. Säätörakennekaavioissa voidaan nimittäin käyttää suurinta osaa AutomationConcepts-, DistributionAndConcurrency- ja DevicesAndResources-aliprofiilien käsitteistä. Esimerkkejä säätörakennekaaviosta on annettu ainakin kuvissa 4.9 ja 4.13. Myös suurin osa muista yllä mainittujen aliprofiilin esittelyissä käytetyistä kuvista esittävät säätörakennekaaviota.

Viimeinen automaatioprofiilin määrittelemistä kaaviotyypeistä on automaatiosekvenssikaavio, jota käytetään automaatiossa yleisten sekvenssien kuvaamiseen. Se on laajennettu UML:n tilakaaviosta (StateDiagram). Tätä kaaviotyyppiä käytetään AutomationConcepts-aliprofiin Sequences-pakkauksen käsitteiden yhteydessä kuvaamaan automaatiojärjestelmän sekventiaalista toiminnallisuutta. Esimerkki kaaviotyypistä on annettu Sequences-pakkauksen esittelyn yhteydessä kuvassa 4.11.

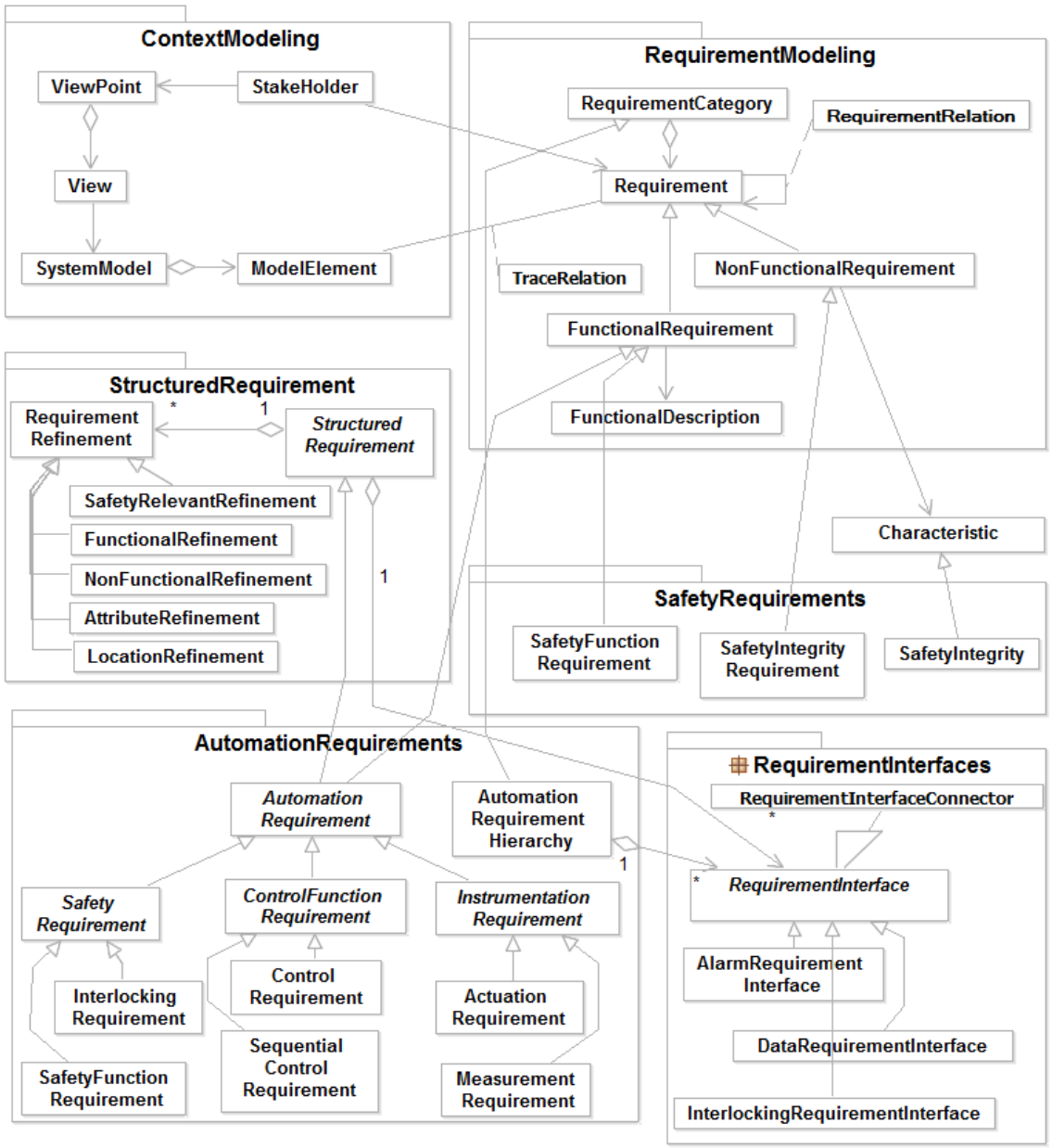
Kuten edellä mainittiin, myös muita UML:n ja SysML:n kaaviotyyppejä voidaan käyttää. Mielekkäitä ovat ainakin UML:n tila- ja komponenttikaavio. Komponenttikaaviolla voidaan kuvata järjestelmän komponenttirakennetta ja komponenttien välisiä rajapintoja. Tilakaaviolla voidaan puolestaan kuvata tiloja, niiden välisiä siirtymiä sekä

siirtymiin liittyviä tapahtumia ja viestejä. Yksinkertainen esimerkki tilakaavion käytöstä automaatioprofiilin kontekstissa on annettu kuvassa 4.12.

4.3.3. Requirements-aliprofiili

Vaatimukset ovat järjestelmän ja sen suunnittelun peruspilari. Vaatimusten koostaminen ja/tai mallintaminen onkin ehkä suunnitteluprosessin kriittisin vaihe, minkä tahansa järjestelmän osalta. Vaatimukset määrittelevät mitä järjestelmän tulee tehdä ja minkä reunaehtojen puitteissa. Kun vaatimukset on saatu aukottomasti koostettua, on suunnitteluprosessin loppuosa näiden vaatimusten täyttämistä ohjelmiston ja laitteiston avulla, jos kyse on teknisestä järjestelmästä. Vaikka tähän ei yleensä päästäkään, sitä tulisi pitää vaatimusmallinnuksen tavoitteena.

Automaatioprofiilissa vaatimukset otetaan huomioon Requirements-aliprofiilissa. Onkin helposti havaittavissa, että Requirements-aliprofiilin käsitteistö edustaa aliluvussa 4.1.1 esitettyä vaatimuskäsitettä. Aliprofiilin käsitteillä on mahdollista luoda hierarkkinen ja jäljitettävä vaatimusmalli, josta muodostuu suunnitteluprosessin lähtökohta. Aliprofiilin käsitteistöä hyödynnetään suunnitteluprosessin alkuvaiheissa. Requirements-aliprofiili koostuu kuudesta pakkauksesta, jotka on esitetty kuvassa 4.4: RequirementModeling, SafetyRequirements, AutomationRequirements, StructuredRequirements, RequirementInterfaces ja ContextModeling.

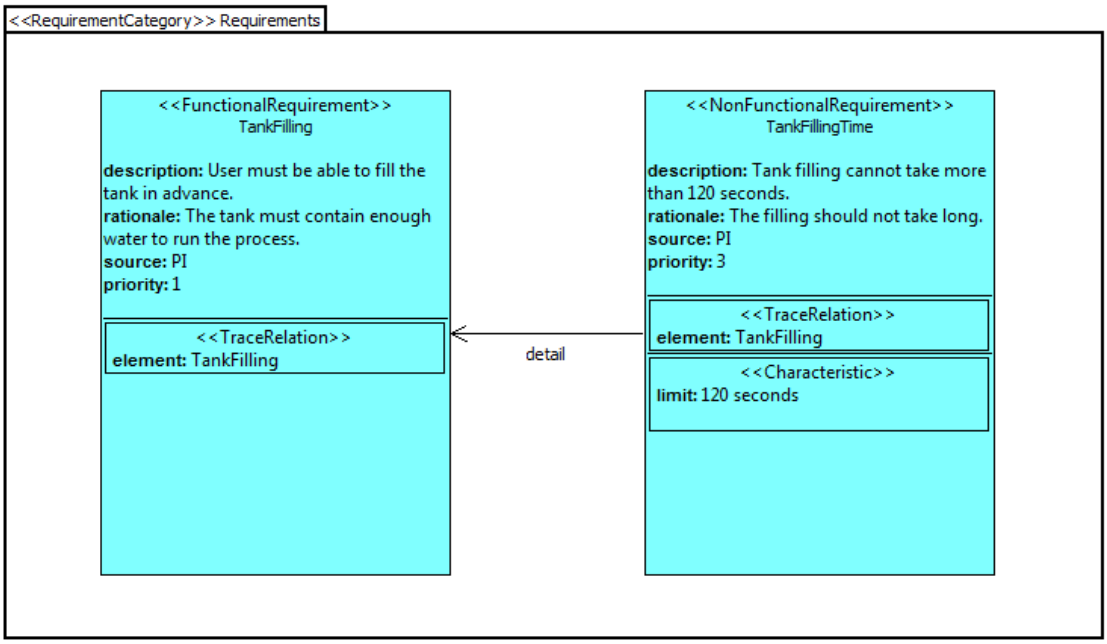


Kuva 4.4. *Requirements-aliprofiilin yleisnäkymä [31]*

RequirementModeling

Käsitteellisesti tärkein pakkaus Requirements-aliprofiilissa on RequirementModeling-pakkaus. Pakkauksen vaatimus-konsepti (Requirement) on koko aliprofiilin kannalta oleellinen käsite, johon kaikki muut varsinaiset vaatimukset pohjautuvat. Pakkauksen käsitteillä on mahdollista mallintaa yleisiä, toiminnallisia ja ei-toiminnallisia vaatimuksia sekä luoda jäljitettävyystietoa vaatimuksista ne toteuttaviin automaatiotoimintoihin (katso AutomationConcepts-aliprofiili 4.3.4), kategorisoida vaatimuksia ja luoda vaatimusten välille relaatioita. Voidaan sanoa, että pakkauksen käsitteillä voidaan mallintaa yleisen tason epäformaaleja vaatimuksia. Käsitteillä voidaan siis luoda ihmisen helposti ymmärtämä, kategorisoitu ja hierarkkinen vaatimusmalli.

Kuvassa 4.5 on esitetty yksinkertainen vaatimusmalli (tai sen osa), joka on luotu RequirementsModeling-pakkauksen käsitteillä. Esimerkissä on yhteen vaatimuskategoriaan (RequirementCategory) määritelty automaatiojärjestelmälle toiminnallinen (FunctionalRequirement) ja sitä tarkentava ei-toiminnallinen (NonFunctionalRequirement) vaatimus, joista ei-toiminnallinen vaatimus on merkitty tarkentamaan toiminnallista vaatimusta. Huomaa myös jäljitysrelaatiot (TraceRelation), jotka linkittävät vaatimukset ne toteuttaviin automaatiotoimintoihin.

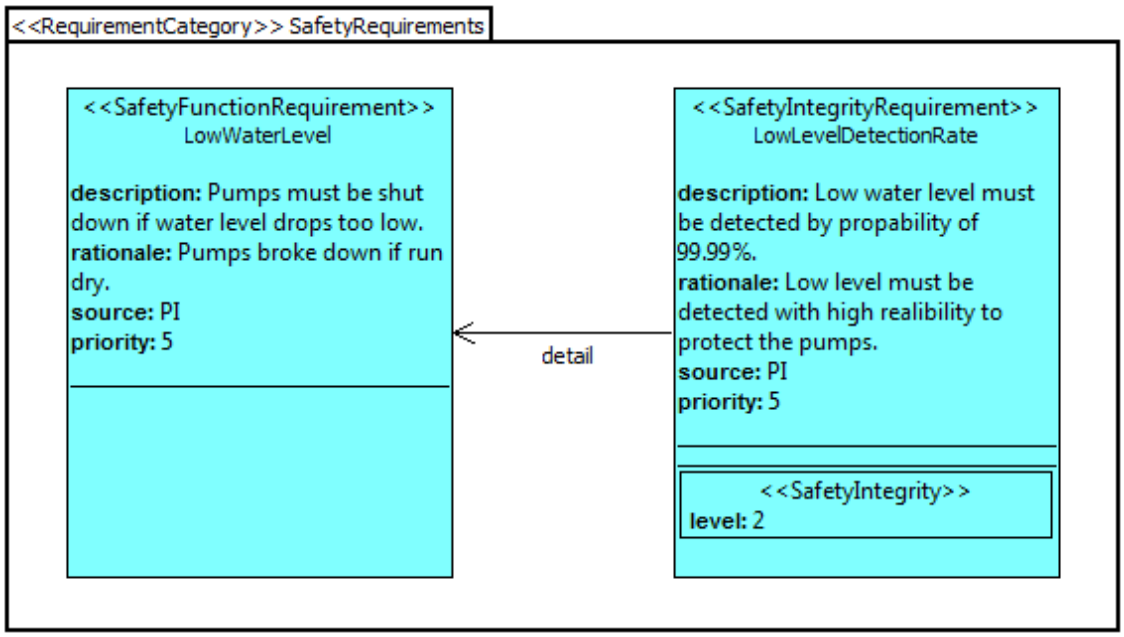


Kuva 4.5. *Esimerkki RequirementModeling-pakkauksen käsitteiden soveltamisesta [30]*

SafetyRequirements

Turvallisuus on automaatiojärjestelmissä keskeisessä asemassa. Vaikka tietty turvallisuustaso onkin usein jo viranomaisten toimesta vaadittu ja vaatimukset olemassa, tulee järjestelmän turvallisuuteen liittyvät vaatimukset määritellä usein vielä muita vaatimuksia tarkemmin. Tätä tarkoitusta varten on automaatioprofiilissa SafetyRequirements-pakkaus. Sen käsitteistöllä mallinnetaan turvallisuuteen liittyviä vaatimuksia. Pakkaus sisältää muun muassa turvatoimintovaatimuksen (SafetyFunctionRequirement) ja turvallisuuden eheysvaatimuksen, jotka on periytetty Requirements-pakkauksen käsitteistöstä. Lisäksi pakkaus sisältää käsitteen, jolla voidaan määritellä eksplisiittisesti vaatimuksen SIL (Safety Integrity Level), eli turvallisuuden eheystaso.

Kuvassa 4.6 on annettu esimerkki turvallisuuteen liittyvien vaatimusten mallintamisesta SafetyRequirements-pakkauksen käsitteillä. Mallissa määritellään turvatoiminto, joka suoritetaan jos tankin pinnankorkeus ajautuu liian matalalle. Toiminnallista vaatimusta tarkennetaan ei-toiminnallisella vaatimuksella, joka tässä tapauksessa edustaa turvallisuuden eheystasoa. Pinnankorkeuden alhaisen tason havaitsemisen todennäköisyydeksi määritellään 99,99%, joka vastaa SIL-tasoa 2, mikä on myös määriteltynä.



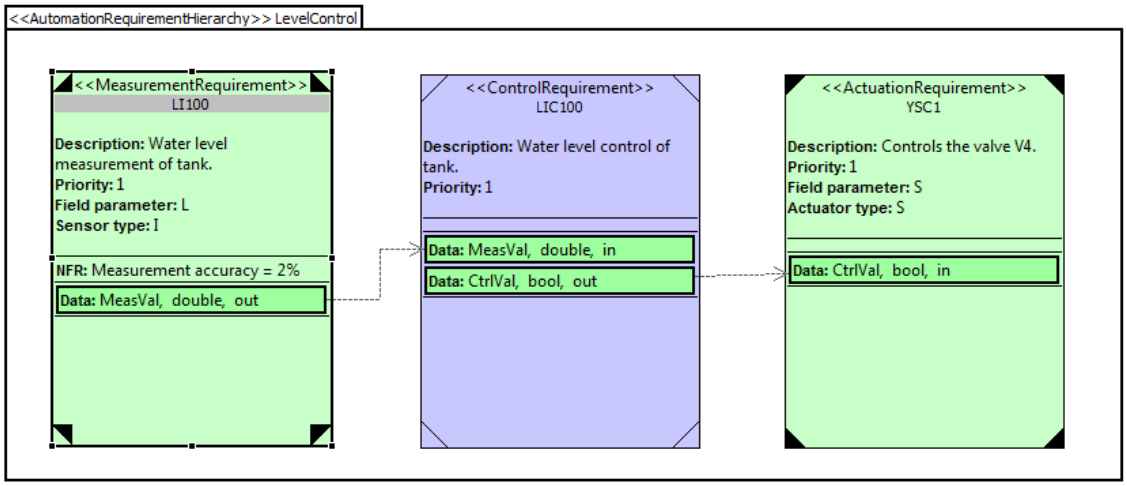
Kuva 4.6. *Esimerkki SafetyRequirements-pakkauksen käsitteiden soveltamisesta [30]*

AutomationRequirements, StructuredRequirements ja RequirementInterfaces

Edellä esitellyt Requirements-aliprofiilin pakkaukset ovat sisältäneet käsitteitä, joilla voidaan mallintaa epäformaaleja vaatimuksia. Epäformaalissa muodossa olevia tekstuaallisia vaatimuksia on kuitenkin hankala käsitellä koneellisesti. On erittäin vaikea kirjoittaa ohjelma, joka vaatimusten tekstuaalisia kuvauksia lukemalla osaa päätellä minkälaisia automaatiotoimintoja tarvitaan vaatimusten toteuttamiseen. Tästä syystä automaatioprofiiliin on sisällytetty myös rakenteisessa muodossa olevia vaatimuskäsitteitä, joita löytyy AutomationRequirements-, StructuredRequirements- ja RequirementInterfaces -pakkauksista. Pakkausten käsitteillä pystytään määrittelemään formaalimmassa muodossa olevaa tietoa ja näin ollen automaattinen käsittely mahdollistuu. Pakkausten vaatimukset on periytetty RequirementModeling-pakkauksen käsitteistä, joten myös ihmisen on helppo tulkita niitä, koska niihin on periytynyt myös tekstuaalinen osa.

AutomationRequirements-pakkaus sisältää muun muassa rakenteiset vastineet RequirementModeling-pakkauksen vaatimukselle (AutomationRequirement) ja kategorialle (AutomationRequirementHierarchy). AutomationRequirement-pakkaus sisältää mittaukselle, säädölle ja ohjaukselle omat rakenteiset vaatimuksensa, joilla kyseisten toimintojen vaatimuksia voidaan esittää. RequirementInterfaces-pakkaukseen kuuluvat muun muassa vaatimusrajapinta (RequirementInterface) ja yhteys (RequirementInterfaceConnector). Vaatimusrajapinnoilla ja niiden välisillä yhteyksillä voidaan määritellä vaatimusten riippuvuuksia informaation (esimerkiksi mittaustiedon) suhteen. StructuredRequirements-pakkaus puolestaan sisältää muun muassa vaatimusten attribuuttitiedon esittämiseen käytetyn elementin (RequirementRefinement). Attribuuteilla voidaan vaatimuksille määritellä formaaleja tai epäformaaleja lisämääreitä, joita tavallisiin vaatimuksiin ei kuulu.

Kuvassa 4.7 on annettu esimerkki AutomationRequirements, StructuredRequirements ja RequirementInterfaces -pakkausten käsitteiden hyödyntämisestä. Esimerkissä on määritelty vaatimuksia tankin pinnankorkeuden säädölle. Esimerkissä on vaatimus pinnankorkeuden mittaamiselle, säätämiselle ja venttiilin ohjaamiselle. Pintamittaus tuottaa mittausinformaatiota, jota säädin hyödyntää ja säädin tuottaa ohjaussignaalin, jota venttiilinohjaus hyödyntää. Vaatimusten kuvaaman informaation perusteella pystytään automaattisesti päättelemään, minkälaisilla automaatiotoiminnoilla vaatimukset voidaan toteuttaa. Päättelyyn käytetään erityisesti vaatimuksiin liitettyjä stereotyyppejä, attribuutteja ja rajapintoja.



Kuva 4.7. *Esimerkki AutomationRequirements, StructuredRequirements ja RequirementInterfaces -pakkauksten käsitteiden soveltamisesta*

ContextModeling

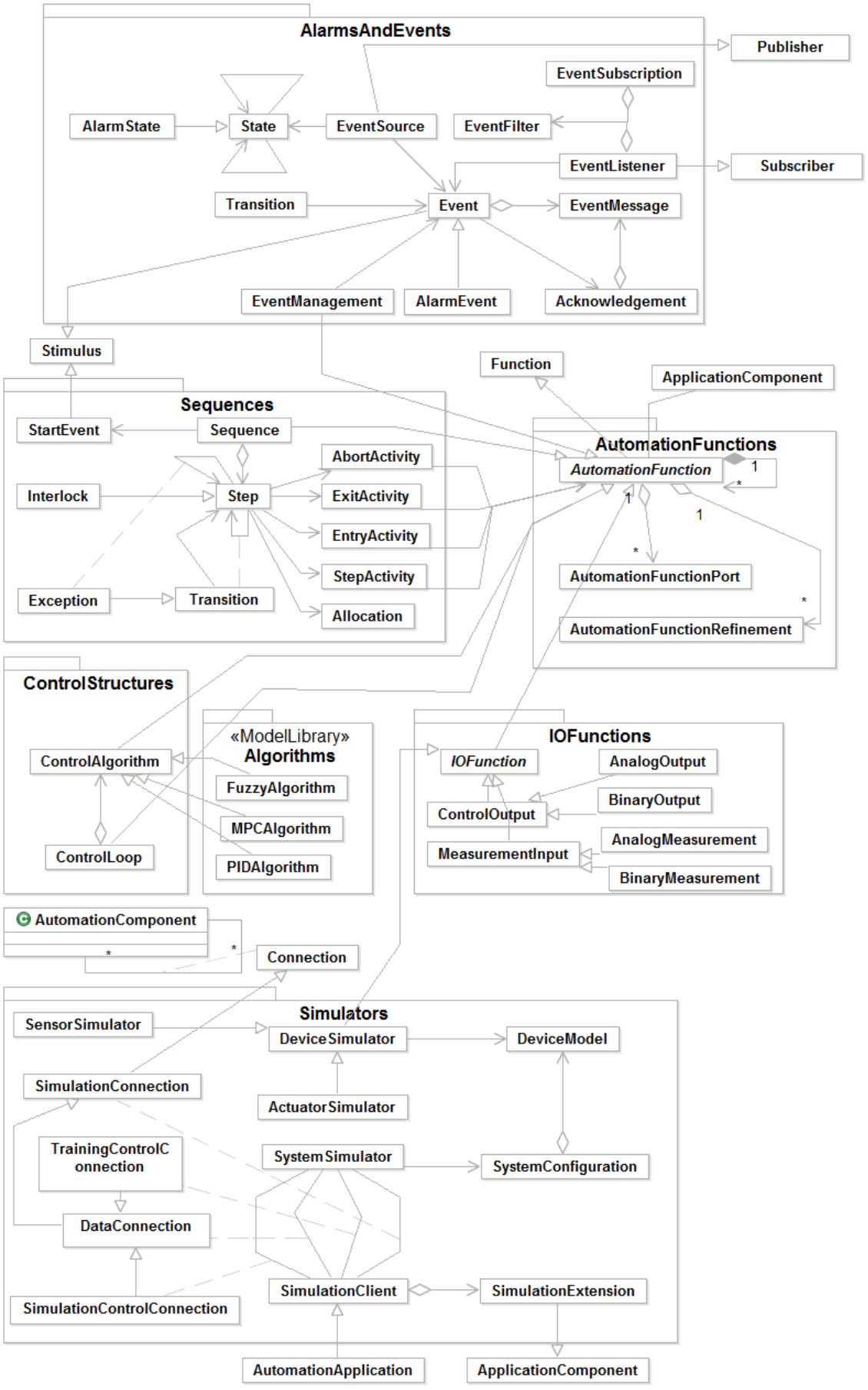
Viimeinen Requirements-aliprofiilin pakkauksista on ContextModeling. Kuten edellä jo kävi ilmi, automaatiosuunnitteluun osallistuu useita suunnitteluosapuolia. Näistä kaikki eivät yleensä ole kiinnostuneita koko järjestelmän mallista vaan jostain sen tietystä osasta. Esimerkiksi asiakas haluaa ehkä nähdä mallista eri osan kuin laite- tai prosessisuunnittelija.

ContextModeling-pakkauksen käsitteillä voidaan luoda erilaisia näkymiä malliin. Tietty osa mallista määritellään kuuluvaksi tiettyyn näkymään ja eri suunnitteluosapuolien näkymiin määritellään heidän kannaltaan mielenkiintoiset osat. Edellä esiteltyjen pakkausten esimerkkeihin viitaten voitaisiin todeta, että RequirementModeling-pakkauksen esimerkki (kuva 4.5) voisi olla asiakasta, SafetyRequirements-pakkauksen esimerkki (kuva 4.6) viranomaista ja AutomationRequirements-pakkauksen esimerkki (kuva 4.7) sovellussuunnittelijaa kiinnostava näkymä malliin.

4.3.4. AutomationConcepts-aliprofiili

Vaikka vaatimukset ovatkin suunnitteluprosessin kulmakivi, ei niillä pystytä järjestelmää kuvaamaan riittävän tarkasti. Tarvitaankin käsitteistöä, jolla automaatiojärjestelmän toiminnallisuutta ja rakennetta voidaan mallintaa tarkemmin, mutta toimialakohtaisen käsitteistön puitteissa. Käsitteistöllä tulee voida mallintaa automaation konsepti, eli se, minkälaisilla automaatiotoiminnoilla vaatimustenmukainen automaatiojärjestelmä saadaan toteutettua.

Automaatioprofiilin toinen aliprofiili on AutomationConcepts-aliprofiili. Se määrittelee yllä olevassa kappaleessa kuvattua käsitteistöä, joka vastaa yleisen käsitteistön automaatiotoimintoja (katso 4.1.2). Aliprofiilin käsitteillä mallinnetaan järjestelmän toiminnallisuus ja rakenne toimialakohtaista käsitteistöä hyödyntäen. Aliprofiili koostuu kuudesta pakkauksesta, jotka ovat: AutomationFunctions, IOFunctions, ControlStructures, Algorithms, Sequences, AlarmsAndEvents ja Simulators. Aliprofiilin pakkausten rakenne ja niiden välisiä riippuvuuksia on esitetty kuvassa 4.8.



Kuva 4.8. *AutomationConcepts-aliprofiilin yleisnäkymä [31]*

AutomationFunctions

Kuten Requirements-aliprofiilissa, myös AutomationConcepts-aliprofiilissa on yksi pakkaus, joka on käsitteellisesti erittäin keskeinen koko pakkauksen kannalta. AutomationConcepts-aliprofiilissa se on AutomationFunctions-pakkaus. Tämä pakkaus on tärkeä siksi, että se sisältää automaatiotoiminto-käsitteen (AutomationFunction). Pakkauksen automaatiotoiminto on abstrakti, mutta siitä periytyy moni aliprofiilin elementeistä, esimerkiksi kaikki IOFunctions-pakkauksen käsitteet.

AutomationFunctions-pakkaus sisältää automaatiotoimintoa kuvaavan käsitteen lisäksi kaksi muuta koko pakkauksen käytön kannalta tärkeää käsitettä, jotka ovat automaatiotoimintoportti (AutomationFunctionPort) ja -tarkenne (AutomationFunctionRefinement). Automaatiotoimintoporttien avulla määritellään yhteyksiä automaatiotoimintojen välille, joiden kautta toiminnot voivat lähettää ja vastaanottaa informaatiota. Portit muodostavat samalla automaatiotoiminnon tarjoaman ja vaatiman rajapinnan. Automaatiotoimintotarkenteella lisätään automaatiotoimintoihin niitä tarkentavaa informaatiota. Automaatiotoimintoja portteineen ja tarkenteineen on käytetty kuvassa 4.9 esitetyn ControlStructures-pakkauksen esimerkin yhteydessä.

IOFunctions

Automaatiojärjestelmässä IO-toiminnot (Input/Output) muodostavat tärkeän osa-alueen. IO-toimintojen tarkoitus on muodostaa yhteys laitteiston ja ohjelmiston välille. Tämän yhteyden kautta automaation sovellusohjelmisto voi ohjata laitteistoa ja saada laitteistolta tarvitsemaansa informaatiota, esimerkiksi mittaussignaaleja. AutomationConcepts-aliprofiilin IOFunctions-pakkaus määrittelee joukon IO-toimintoja mallintavia käsitteitä, joilla voidaan mallintaa mittaussignaalien vastaanottamista ja toimilaitteiden ohjausta.

IOFunctions-pakkaus muodostuu abstrakteista pääkäsitteistä, joita ovat mittaussisääntulo (MeasurementInput) ja ohjauslähtö (ControlOutput). Nämä toiminnot toteuttavat abstraktilla tasolla mittaussignaalien lukemisen ja laitteiden ohjauksen. Pakkauksen konkreettiset käsitteet periytyvät mainituista abstrakteista käsitteistä ja ne toteuttavat analogisen ja binäärisen version mittaussisääntulosta ja ohjauslähdöstä (BinaryMeasurement, AnalogMeasurement, BinaryOutput ja AnalogOutput). Kuvassa 4.9 on esitetty esimerkit AnalogMeasurement- ja BinaryOutput-elementeistä, jotka on täydennetty AutomationFunctions-pakkauksen porteilla ja tarkennuksilla.

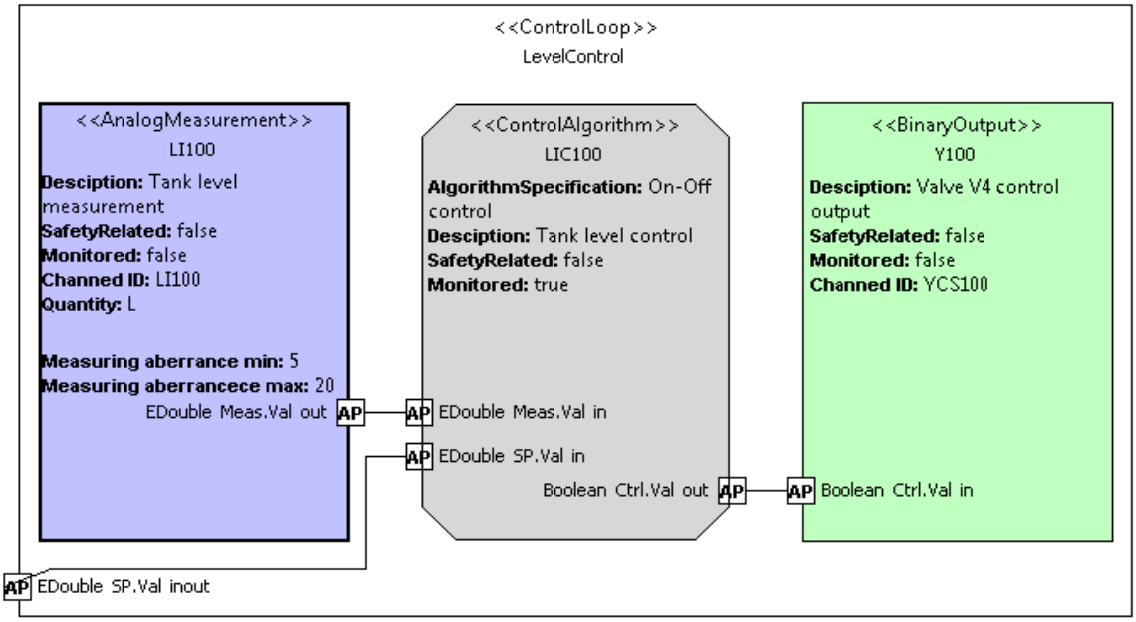
ControlStructures

Automaatiojärjestelmän ehkä yleisin toiminto on säätöpiirin suorittaminen. Säätöpiiri määrittelee piirin, joka (yleensä) mittaa säädettävää suuretta laskee ohjauksen ja lähettää sen toimilaitteelle, joka pystyy vaikuttamaan säädettävään suureeseen. Säätöpiiri voi olla hyvin yksinkertainen SISO-säädin (Single-Input Single-Output), jossa yhden mittauksen perusteella säädetään yhtä suuretta. Säätöpiiri voi olla myös hyvin monimutkai-

nen MIMO-säädin (Multi-Input Multi-Output), jossa monen mittauksen perusteella säädetään useaa suuretta.

ControlStructures-pakkaus sisältää geneeriset säätöpiiri (ControlLoop) ja säätöalgoritmi (ControlAlgorithm) käsitteet. Säätöpiiri rajaa yhden piirin toiminnallisuuden erilleen muista mahdollisista säätöpiireistä, eli sen sisällä on määritelty kyseisen piirin rakenne. Säätöalgoritmi-elementti mallintaa geneeristä säätöalgoritmia, johon käyttäjä voi itse määritellä algoritmin tekstuaalisesti. AutomationConcepts-aliprofiilista löytyy myös Algorithms-pakkaus, joka määrittelee erikoistuneempia säätöalgoritmeja.

Kuvassa 4.9 on annettu esimerkki ControlStructures-pakkauksen käsitteiden soveltamisesta. Kuvassa on määritelty hyvin yksinkertainen pinnansäätöpiiri pakkauksen käsitteitä hyödyntäen. Piirissä on hyödynnetty myös AutomationFunctions-pakkauksen käsitteitä (kuvassa <<AnalogMeasurement>> ja <<BinaryOutput>>).



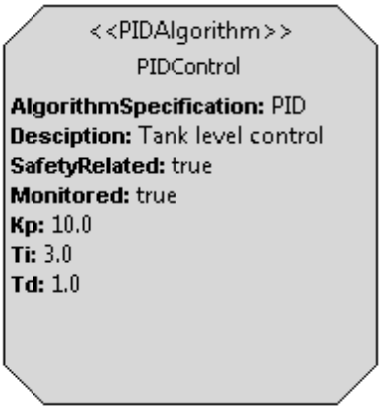
Kuva 4.9. *Esimerkki ControlStructures-pakkauksen käsitteiden soveltamisesta (täydennettynä AutomationFunctions-pakkauksen käsitteillä)*

Algorithms

ControlStructures-pakkauksen säätöalgoritmikäsite on, kuten mainittiin, hyvin yleisellä tasolla oleva käsite, joka kattaa kaikentyyppiset säätöalgoritmit. Automaatiossa kuitenkin yleensä käytetään esimerkiksi PID- ja sumeita säätimiä. Algorithms-pakkaus sisältää kolme erikoistunutta säädintyyppiä jotka ovat PID (Proportional-Integral-Derivative), sumea ja MPC (Model Predictive Control). PID-elementti (PIDAlgorithm) mallintaa tavanomaista PID-algoritmia ja siinä on käyttäjän aseteltavissa olevat vahvistus, integrointi- ja derivointiaika parametrit. Sumea algoritmi (FuzzyAlgorithm) edustaa mitä tahansa sumeaa säätöalgoritmia ja MPC-algoritmi (MPCAlgorithm) puolestaan mitä tahansa mallipredikti ivistä säätöalgoritmia. Sumeaa ja mallipredikti ivistä säätöalgorit-

mia ei pystytä parametroimaan samalla tavalla kuin PID-elementtiä, koska niillä ei ole PID-säädintä vastaavia parametreja.

Kaikkia Algorithms-pakkauksen elementtejä voidaan käyttää yleisen säätöalgoritmielementin tilalla, jos käytettävän säätimen tyyppi on ennalta tiedossa. Eli esimerkiksi kuvassa 4.9 esitetyn säätöpiirin yleinen säätöalgoritmi voitaisiin korvata millä tahansa Algorithms-pakkauksen elementeistä, mikäli näin haluttaisiin. Kuvassa 4.10 on annettu esimerkki PID-algoritmielementin ilmentymästä UML AP-työkalussa.



Kuva 4.10. *PIDAlgorithm-elementin graafinen ilmentymä UML AP-työkalussa*

Sequences

Sekvenssi on tärkeä käsite, joka on automaatiossa yleinen esimerkiksi panosautomaation alueella. Sekvenssillä voidaan kuvata esimerkiksi prosessin etenemisen vaiheita tai yleisesti mitä tahansa vaiheittaista ja mahdollisesti samanaikaista toimintaa. Automaatioprofiilissa sekvenssien mallintaminen on huomioitu Sequences-pakkauksessa.

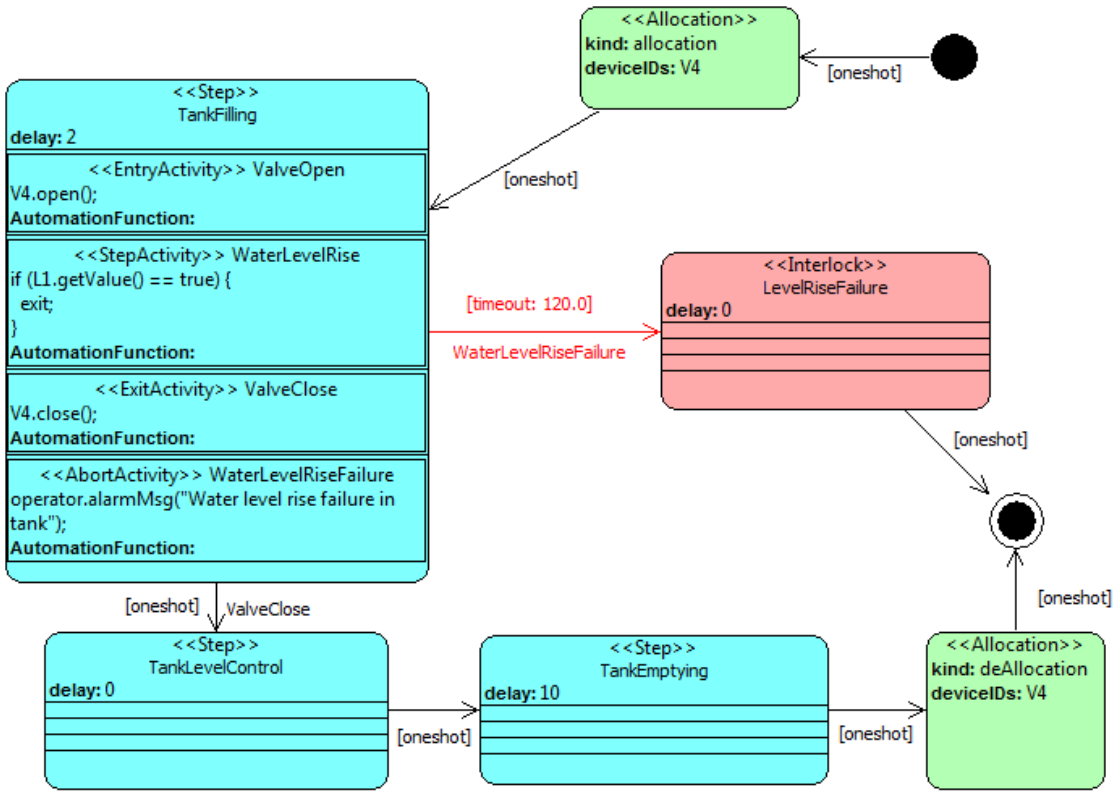
AutomationConcepts-aliprofiilin Sequences-pakkaus sisältää sekvenssien kuvaamiseen tarvittavaa käsitteistöä. Sekvensseillä voidaan automaatioprofiilin kontekstissa kuvata esimerkiksi prosessien tai säätöpiirien suoritusta. Pakkauksen ylimmän tason käsite on sekvenssi (Sequence), joka on automaatiotoiminnosta periytetty automaatiojärjestelmän toiminnallisuus. Sekvenssit koostuvat askelista (Step), siirtymistä (Transition) ja näiden erityismuodoista, kuten lukituksista (Interlock) ja poikkeuksista (Exception). Pakkaus sisältää myös allokointi-elementin, jolla sekvenssi voi varata ja vapauttaa laitteita tai resursseja käyttöönsä.

Sekvenssin pääketju kuvataan käyttäen askeleita ja niiden välisiä siirtymiä. Mahdolliset poikkeustilanteet taas mallinnetaan lukituksia ja poikkeuksia hyödyntäen. Askeleet ja lukitukset koostuvat vielä neljästä sisäisestä aktiviteetista, joilla voidaan mallintaa askeleen tuloaktiviteetti (EntryActivity), pääaktiviteetti (StepActivity), poistumisaktiviteetti (ExitActivity) ja keskeytysaktiviteetti (AbortActivity). Tuloaktiviteetissa suoritetaan yleensä mahdolliset alustukset ja tarkistukset, joita tarvitaan myöhemmissä aktiviteeteissa. Pääaktiviteetissa suoritetaan askeleen varsinaiset toimenpiteet. Poistumisaktiviteetissa voidaan esimerkiksi saattaa ohjattava prosessi tai sen osa askeleen jälkeen

sopivaan tilaan. Keskeytysaktiviteetti suoritetaan yleensä erilaisissa vikatilanteissa, joten siinä voidaan yrittää toipua vikatilanteesta tai lähettää hälytyksiä.

Kuvassa 4.11 on esimerkki Sequences-pakkauksen käsitteiden soveltamisesta. Kaaviossa käsitteiden avulla määritellään yksinkertainen kolmivaiheinen sekvenssi, jossa vesitankki täytetään, tankin pinnankorkeutta säädetään ja lopuksi tankki tyhjennetään. Sekvenssiin kuuluu varsinaisten askeleiden lisäksi myös prosessilaitteiden varaaminen (Allocation) ja yksi lukitusaskel.

Kuvassa 4.11 äärimmäisenä vasemmalla olevan askeleen (TankFilling) aktiviteeteissa on esitetty pseudokoodia, jolla on tarkoitus esittää kunkin vaiheen toiminnallisuutta. Tämä ei ole aktiviteettien suositeltu käyttötapa, vaikka automaatioprofiili salliikin esitetyn kaltaisen mallintamisen. Normaalisti aktiviteetteihin liitettäisiin automaatiotoiminto tai automaatiotoimintoja, jotka suoritettaisiin kyseisen aktiviteetin aikana. Tässä esimerkissä on kuitenkin haluttu havainnollistaa aktiviteettien toiminnallisuutta pseudokoodin kautta. Askelaktiviteetti (WaterLevelRise) suoritetaan sadan millisekunnin välein kunnes kytkimen L1 taso saavutetaan (suoritusväli ei ilmene kuvasta).



Kuva 4.11. *Esimerkki Sequences-pakkauksen käsitteiden soveltamisesta [30]*

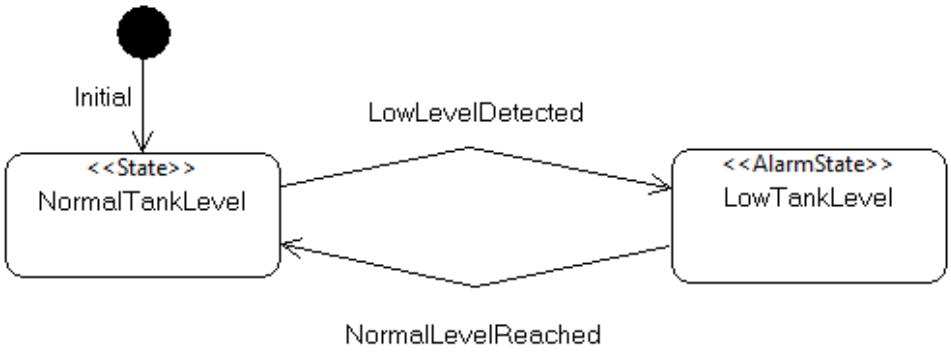
AlarmsAndEvents

Tapahtuma on tärkeä käsite (automaation)ohjelmistotekniikassa. Se voi viitata esimerkiksi napin painallukseen, painerajan ylittymiseen tai virta-anturin signaaliin, joka merkitsee virtauksen pysähtymistä. Tapahtuman ilmeneminen tarkoittaa yleensä, että oh-

jelmiston tulee reagoida jollain tavalla. Ohjelmisto voi esimerkiksi vaihtaa tilaansa tai käynnistää jonkin tietyn ikkunan käyttäjän tarkasteltavaksi.

Automaatioprofiilissa tapahtumia ja ohjelmiston tilaa mallinnetaan AlarmsAndEvents-pakkauksen käsitteillä. Tapahtumien mallintamisen oleellisin käsite on tapahtuma (Event), josta on olemassa myös hälytyksiin erikoistettu hälytys (AlarmEvent). Pakkaus sisältää tapahtumiin liittyen myös käsitteistöä, jolla voidaan mallintaa tapahtumien yhteydessä lähetettäviä viestejä (EventMessage) sekä näiden lähettämiseen, vastaanottamiseen ja suodattamiseen liittyvää käsitteistöä. Toinen tärkeä pakkauksen vastuualue on tilan mallintaminen. Tässä keskeinen käsite on tila (State), jolla kuvataan ohjelmiston tila. Tilojen välisiä siirtymiä (Transition) ja hälytystiloja (AlarmState) on myös mahdollista määritellä.

Tapahtumat ovat tiukasti sidoksissa järjestelmän tiloihin, koska tapahtumia generoituu tilojen välisissä siirtymissä. Kuvassa 4.12 on annettu esimerkki AlarmsAndEvents-pakkauksen käsitteiden käytöstä. Esimerkissä on määritelty ohjelmistolle kaksi tilaa, jossa se voi olla. Ensimmäisessä (NormalTankLevel) indikoi normaalia tilaa, jossa tankissa on riittävästi nestettä. Toisessa tiloista (LowTankLevel) pinnankorkeus on liian matala, joten tila on itse asiassa hälytystila. Tilojen välisiin siirtymiin liittyy tapahtumia ja tapahtumaviestejä.



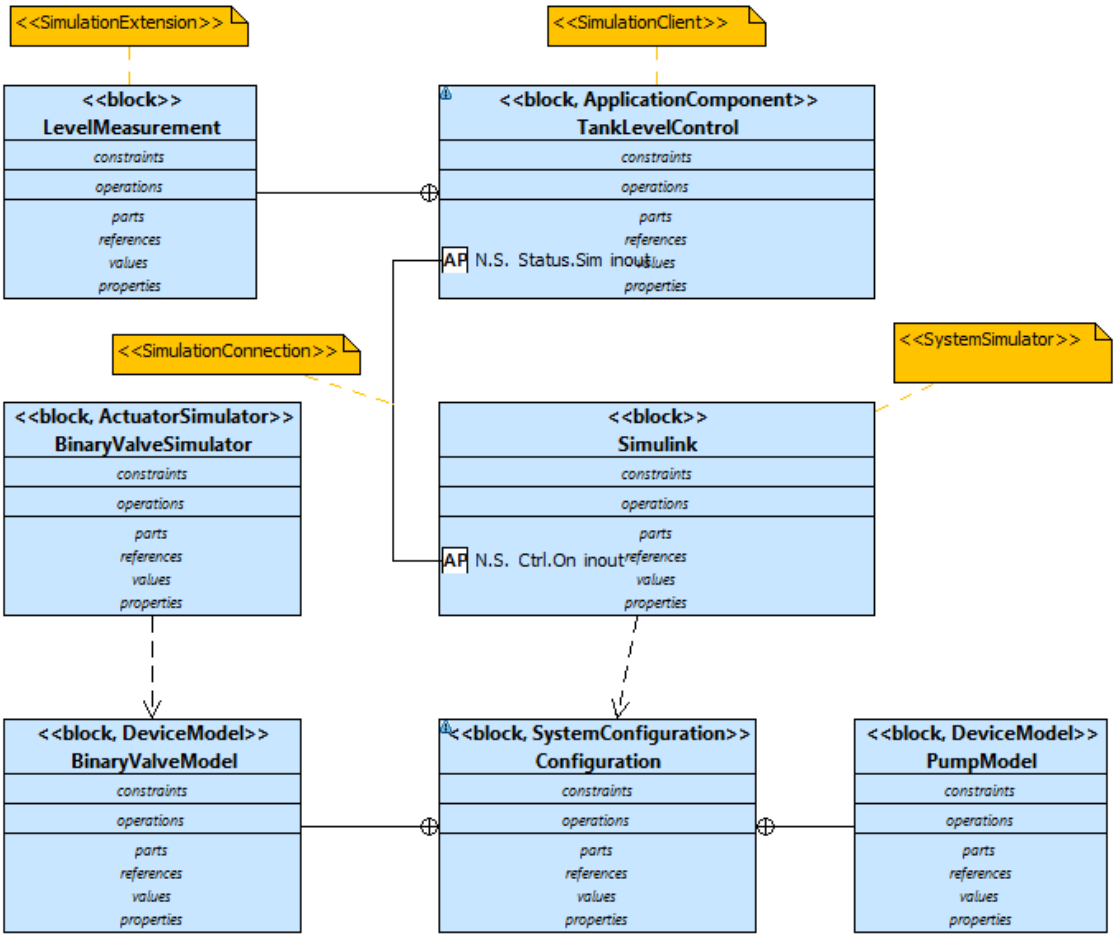
Kuva 4.12. *Esimerkki AlarmsAndEvents-pakkauksen käsitteiden soveltamisesta*

Simulators

Viimeinen AutomationConcepts-aliprofiilin pakkauksista on Simulators-pakkaus. Se koostuu nimensä mukaisesti simulointiin liittyvästä käsitteistöstä. Simulointi on automaatiossa yleisesti hyödynnetty menetelmä muun muassa järjestelmän testaamiseen, virittämiseen ja jossain määrin myös ajonaikaiseen operointiin. Ajonaikaisella operoinnilla viitataan esimerkiksi sensorisimulaattorien käyttöön. Sensorisimulaattori voi muun muassa epäsuorasti approksimoida mittaussuuretta sillä aikaa, kun varsinainen sensori on epäkunnossa tai huollettavana. Näin saadaan säätimelle edes välttävä arvio säädettävän suureen tilasta.

Simulators-pakkaus koostuu käsitteistä, joilla voidaan mallintaa simulaattoreita ja yhdistää niitä muuhun ohjelmistoon. Pakkauksella ei ole yhtä keskeistä käsitettä, jonka

ympärille loput rakentuvat, mutta sen keskeiset osat ovat laitesimulaattorit (DeviceSimulator), järjestelmäsimulaattorit (SystemSimulator) ja simulaatioyhteydet (SimulationConnection). Kuvassa 4.13 on annettu esimerkki Simulators-pakkauksen käsitteiden soveltamisesta. Esimerkissä on määritelty automaatiokomponentin ilmentymä TankLevelControl-järjestelmäkomponentti (ApplicationComponent, esiintyy aliluvussa 4.3.5), joka on simulaattoreiden kannalta asiakas (SimulationClient). Asiakas käyttää järjestelmäsimulaattoria (SystemSimulator), joka edelleen hyödyntää järjestelmän konfiguraatiota (SystemConfiguration), joka taas koostuu laitemalleista (DeviceModel). Laitemalleja käyttävät myös laite- (ActuatorSimulator) ja sensorisimulaattorit (SensorSimulator).



Kuva 4.13. *Esimerkki Simulators-pakkauksen käsitteiden soveltamisesta (mukailtu lähteestä [30])*

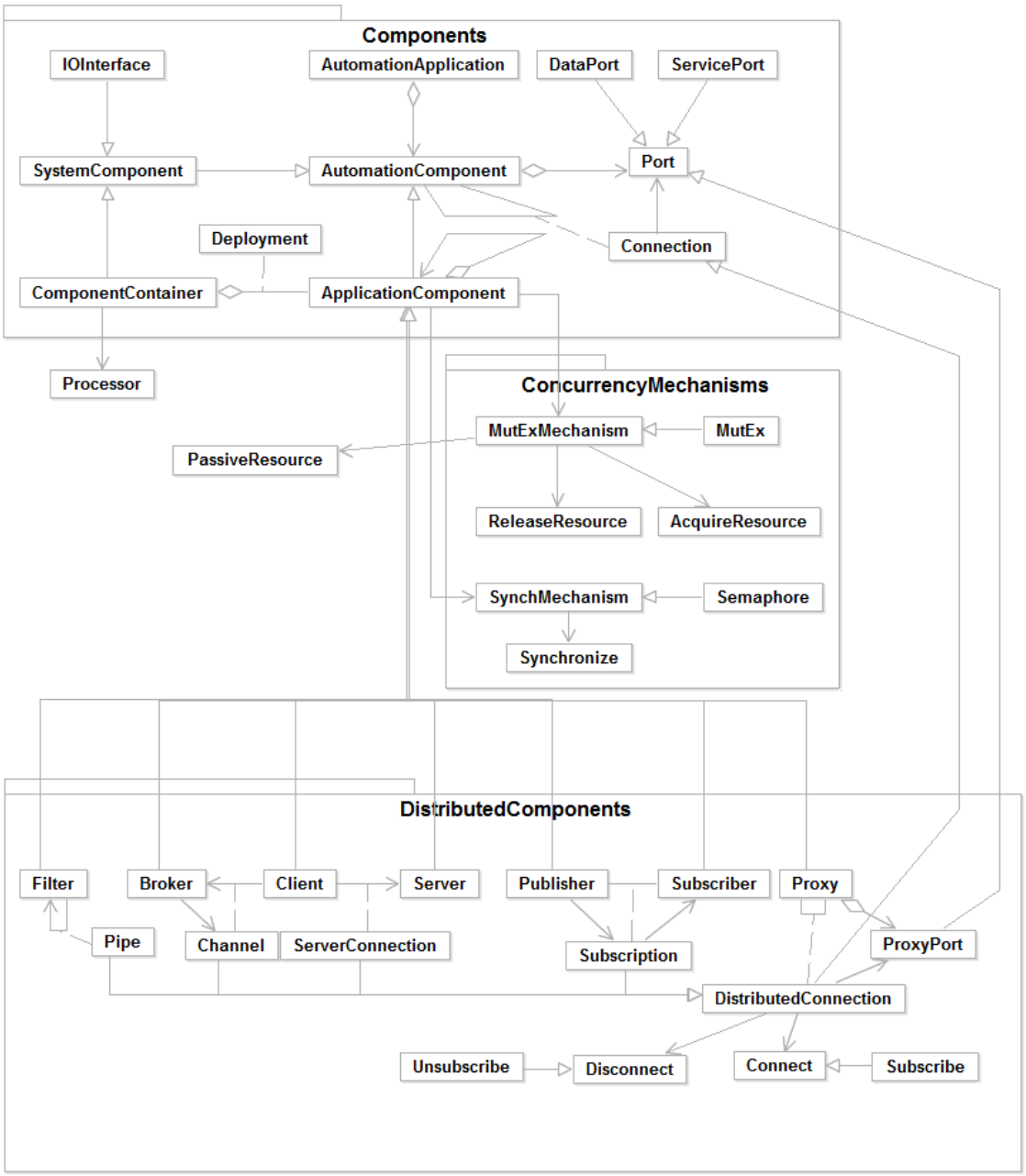
4.3.5. DistributionAndConcurrency-aliprofiili

Automaatiojärjestelmien ollessa nykyisin laajoja ja monimutkaisia systeemejä, ne yleensä toteutetaan hajautettuina järjestelminä. Hajautetussa järjestelmässä ohjelmisto jaetaan osiin, jotka suoritetaan hajautetuissa prosessointiyksiköissä. Tässä vaiheessa tehdään myös hajautukseen liittyviä suunnittelumallien mukaisia suunnitteluratkaisuja,

jotka yleensä lisäävät ohjelmiston komponenttien ja olioluokkien määrää. Ohjelmistoa ajetaan usealla pienemmällä prosessointiyksiköllä yhden massiivisen yksikön sijaan. Tällä saavutetaan etuja, mutta lähestymistavalla on myös ongelmansa. Näitä ongelmia ovat muun muassa rinnakkaisuus ja toimintojen samanaikaisuus, jotka hajautuksen yhteydessä tulevat osaksi ongelmakenttää[4].

Automaatioprofiilin DistributionAndConcurrency-aliprofiili sisältää käsitteistöä, jolla voidaan mallintaa järjestelmän hajautusta komponentteihin, hajautettujen komponenttien kommunikointimalleja ja rinnakkaisuusmekanismeja. Aliprofiili koostuu Components-, ConcurrencyMechanisms-, ja DistributedComponents-pakkauksista.

[4] Automaatiosuunnittelun kannalta erityisesti toimintojen samanaikaisuus on otettava huomioon. Rinnakkaisuuteen liittyvät ongelmat on mahdollisesti ratkaistu jo kehitysympäristön tai DCS-järjestelmän tasolla, eikä suunnittelijan tällöin tarvitse ottaa niihin kantaa. Automaatiojärjestelmä, jossa rinnakkaisuuteen ja toimintojen samanaikaisuuteen liittyviä kysymyksiä ei tarvitsisi ottaa ollenkaan huomioon, on todella yksinkertainen, koska mainitut ongelmat tulevat esiin jo tapahtumapohjaisessa ohjelmisto- tai käyttöjärjestelmäarkkitehtuurissa, joka on erittäin yleisesti käytetty.



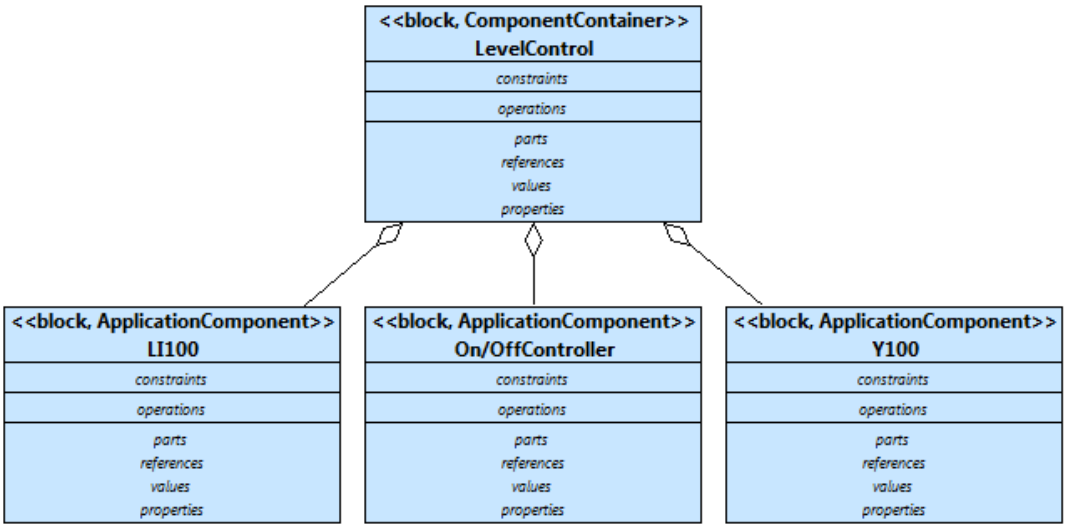
Kuva 4.14. *DistributionAndConcurrency-aliprofiilin yleisnäkymä [31]*

Components

Hajautuksen perustana on järjestelmän jakaminen sopiviin komponentteihin. Komponentti on ohjelmistotekniikan (karkea) vastine elektronisille komponenteille. Elektronisista komponenteista, kuten vastuksista, kondensaattoreista ja keloista pystytään rakentamaan järjestelmiä kytkemällä niitä yhteen. Samalla tavalla ohjelmistokomponenteista voidaan rakentaa järjestelmiä kytkemällä komponentteja yhteen. Tosin ohjelmistopuolella komponenttien kytkeminen ei ole yhtä suoraviivaista, esimerkiksi monimutkaisempien rajapintojen takia. Komponenttien käyttö ohjelmistotekniikassa on kuitenkin kasvusuunnassa ja niitä hyödynnetään erityisesti erilaisissa komponenttiympäristöissä (esimerkiksi EJB), jotka tarjoavat komponenteille ajoalustan.

DistributionAndConcurrency-aliprofiilin Components-pakkaus sisältää käsitteistöä, jolla voidaan mallintaa automaatiojärjestelmän komponenttirakennetta ja komponenttien välisiä yhteyksiä. Pakkauksen keskeinen käsite on automaatiokomponentti (AutomationComponent), josta muut pakkauksen käsitteet on periytetty tai johon ne liittyvät. Automaatiokomponentista periytettyjä käsitteitä ovat muun muassa järjestelmäkomponentti (SystemComponent), sovelluskomponentti (ApplicationComponent) ja komponenttisäilö (ComponentContainer), jolle automaatiokomponentteja voidaan allokoida ajettavaksi. Näillä käsitteillä voidaan mallintaa niin ajoalustan tarjoamia (komponentti)palveluita kuin sovelluksen sisäistä komponenttirakennettakin. Lisäksi pakkauksessa määritellään komponenttien väliseen tiedonsiirtoon tarkoitettuja käsitteitä, joita ovat muun muassa dataportti (DataPort), palveluportti (ServicePort) ja yhteys (Connection). Porttien läpi komponentit voivat vaihtaa informaatioita, joka kuljetetaan yhteyksiä pitkin.

Kuvassa 4.15 on annettu esimerkki Components-pakkauksen käsitteiden soveltamisesta. Esimerkissä on esitetty komponenttisäilö (LevelControl) sekä ne sovelluskomponentit (ApplicationComponent), jotka kuuluvat kyseiseen komponenttisäilöön. LevelControl-komponenttisäilö voidaan allokoida ajettavaksi esimerkiksi säätimelle, jolloin siihen kuuluvat sovelluskomponentit suoritetaan kyseisellä säätimellä.



Kuva 4.15. *Esimerkki Components-pakkauksen käsitteiden soveltamiesesta (mukailtu lähteestä [30])*

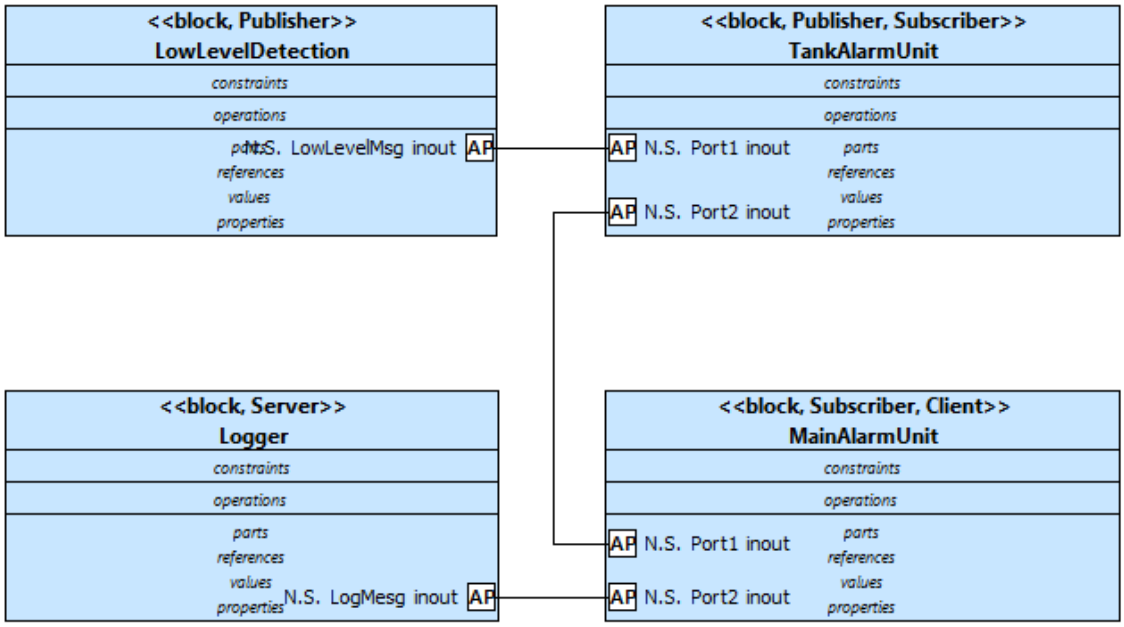
DistributedComponents

Komponenttien yhteydessä hyödynnetään usein jonkinlaista suunnittelumallia. Suunnittelumallit ovat hyväksi havaittuja ratkaisuja niihin ongelmiin, joihin suunnittelumallit on alun perin kehitetty. Automaatioprofiilin kontekstissa suunnittelumalleja on hyödynnetty komponenttien kommunikoinnin toteuttamiseen. DistributedComponents-pakkaus sisältää käsitteitä komponenttien välisten kommunikointimallien esittämiseen. Pakkaus sisältää tarvittavat käsitteet seuraavien kommunikointimallien esittämiseen: proxy (pro-

xy), asiakas-palvelin (client-server), välittäjä (broker), tarkkailija (observer) sekä putket ja suodattimet (pipes and filters).

Kullakin kommunikointimalleista on tyypillinen käyttökohteensa. On kuitenkin yleistä, että käyttökohteeseen sopisi periaatteessa useakin pakkauksen sisältämistä malleista. Tällöin suunnittelija joutuu tekemään päätöksen käytettävästä kommunikointimallista. Profiilille tuotettu ohjeistus ei ota kantaa sopivan kommunikointimallin käyttöön sinänsä, mutta esittelee kunkin suunnittelumallin lyhyesti ja pyrki näin antamaan suunnittelijalle eväitä sopivan suunnittelumallin valintaan. On kuitenkin suositeltavaa tutustua suunnittelumalleihin ja niiden käyttöön myös asiaa käsittelevän kirjallisuuden kautta.

Kuvassa 4.16 on annettu esimerkki DistributedComponents-pakkauksen käsitteiden soveltamisesta. Esimerkissä on käytetty sekä asiakas-palvelin ja tarkkailija -kommunikointimalleja. Esimerkissä on toteutettu monitasoinen hälytyskomponenttien järjestelmä, jonka tiedot tallentuvat lokiin. Ketju alkaa LowLevelDetection-komponentista, joka havaitsee matalan pinnan tason esimerkiksi tankissa. Se on tarkkailijamallin mukainen julkaisija (Publisher) joka lähettää tiedon matalan pinnan ilmenemisestä tilaajille (Subscriber). TankAlarmUnit-komponentti on tässä tilaaja, joka on tilannut tiedon matalan tason ilmaantumisesta. Toisaalta TankAlarmUnit on julkaisija MainAlarmUnit-komponentille. Yhdellä komponentilla voi siis olla erilaisia rooleja erilaisissa kommunikointimalleissa. MainAlarmUnit on siis tilaaja, mutta se on myös asiakas (asiakas-palvelin -mallissa), joka joko määrävälein tai tarvittaessa Logger-komponentin palvelua hyödyntäen tallentaa saapuneet hälytykset lokiin.



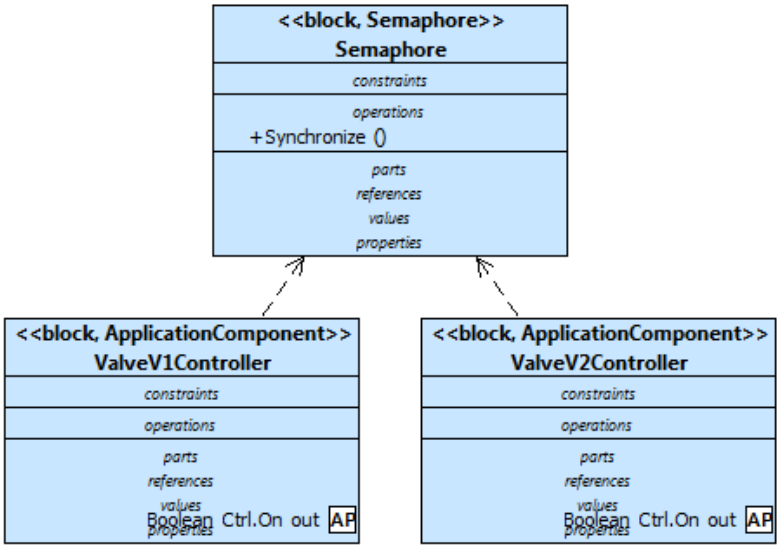
Kuva 4.16. *Esimerkki DistributedComponents-pakkauksen käsitteiden soveltamisesta (mukailtu lähteestä [30])*

ConcurrencyMechanisms

Kuten jo aliluvun alustuksessa mainittiin, toimintojen samanaikaisuudesta ja rinnakkaisuudesta aiheutuvat ongelmat ovat läsnä varsinkin hajautetuissa järjestelmissä, ja näin ollen lähes kaikissa automaatiojärjestelmissä. DistributionAndConcurrency-aliprofiilin ConcurrencyMechanisms-pakkaus ottaa kantaa rinnakkaisuus- ja samanaikaisuusongelmien ratkaisemiseen automaatioprofiilin avulla. Pakkaus sisältää käsitteitä synkronointi- ja resurssien jako-ongelmien ratkaisemiseen. Synkronointiongelmassa komponenttien suoritus halutaan synkronoida siten, että kunkin komponentin suoritus ajoittuu haluttuun aikaikkunaan. Synkronoinnilla voidaan esimerkiksi ajoittaa kahden eri komponentin ohjaaman venttiilin yhtäaikainen aukeaminen. Resurssien jako-ongelmassa on puolestaan kyse siitä, että kaksi tai useampi eri komponenttia ei voi käyttää yhtä fyysistä tai loogista resurssia yhtäaikaisesti. Esimerkiksi yksi venttiili tulee olla kerrallaan vain yhden komponentin ohjauksessa.

Pakkaus sisältää kaksi ohjelmistotekniikan puolelta tuttua käsitettä, joilla voidaan ratkaista edellä esiteltyjä ongelmia. Semaforilla (Semaphore) voidaan ratkoa synkronointiongelmia. Semaforissa on synkronointitoiminto (Synchronize), jota synkronoitavat komponentit kutsuvat. Kun synkronointiehto täyttyy, semafori päästää sitä kutsuneet komponentit jatkamaan. Resurssien jako -ongelmaa varten pakkaus sisältää Mutex-elementin. Mutexilla voidaan rajoittaa tietyn resurssin yhtäaikaista käyttöä. Jotta komponentti voisi käyttää resurssia, sen tulee ensin kutsua sitä vartioivaa mutexia. Jos resurssi on varattu, mutex estää komponentin pääsyn resurssiin.

Kuvassa 4.17 on annettu esimerkki ConcurrencyMechanisms-pakkauksen käsitteiden soveltamisesta. Esimerkissä esitetään semaforin käytön periaate automaatioprofiilin avulla. Esimerkin tilanteessa kaksi venttiilinohjainta haluaa synkronoitua keskenään ja ne käyttävät tähän semaforin synkronointitoimintoa.

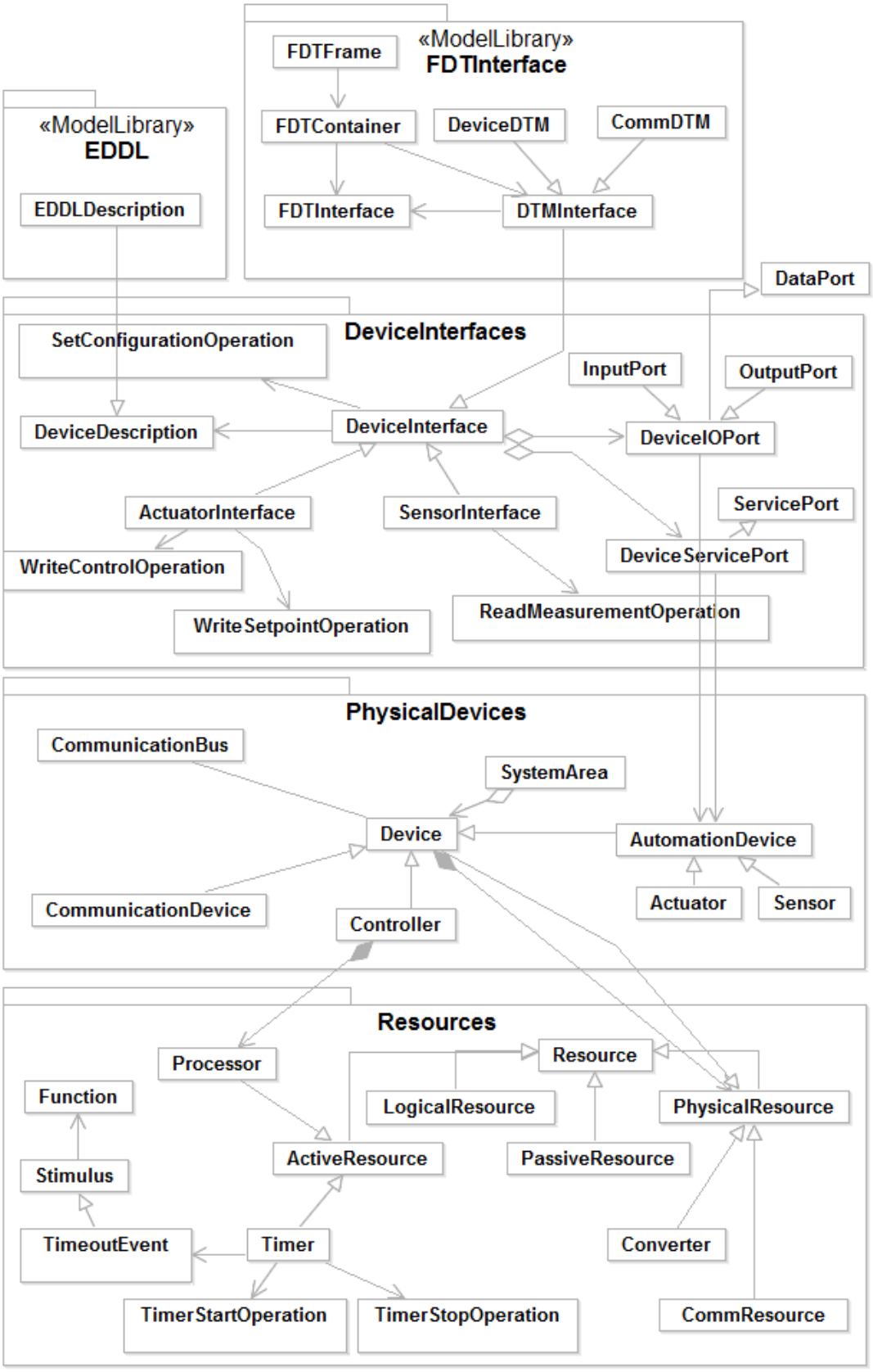


Kuva 4.17. *Esimerkki ConcurrenceMechanisms-pakkauksen käsitteiden soveltamisesta (mukailtu lähteestä [30])*

4.3.6. DevicesAndResources-aliprofiili

Laitteisto ja sen tarjoamat resurssit ohjelmiston kannalta ovat oleellinen osa automaatiojärjestelmää. DevicesAndResources-aliprofiili sisältää käsitteistöä, jolla voidaan mallintaa automaatiojärjestelmän laitteistoa ja laitteistorajapintoja. Aliprofiilin käsitteistö ottaa lisäksi kantaa järjestelmän resurssien sekä laitteiston ja ohjelmiston välisen rajapinnan esittämiseen. Ohjelmiston kannalta laiterajapinta on erittäin oleellinen osamalli, koska sen kautta päästään käsiksi laitteen tarjoamiin toimintoihin ja palveluihin. Resursseilla puolestaan voidaan mallintaa ohjelmiston kannalta erilaisia resursseja, joita voivat olla muun muassa laitteisto tai suoritusyksiköiden loogiset resurssit, kuten massamuisti.

DevicesAndResources-aliprofiilin pakkaukset ja niiden käsitteet on esitetty kuvassa 4.18. Aliprofiili koostuu viidestä pakkauksesta, joista Resources keskittyy resurssien, PhysicalDevices fyysisten laitteiden ja DeviceInterfaces-, FDTInterfaces- sekä EDDL-pakkaukset laiterajapintojen ja -kuvausten käsitteistöön.



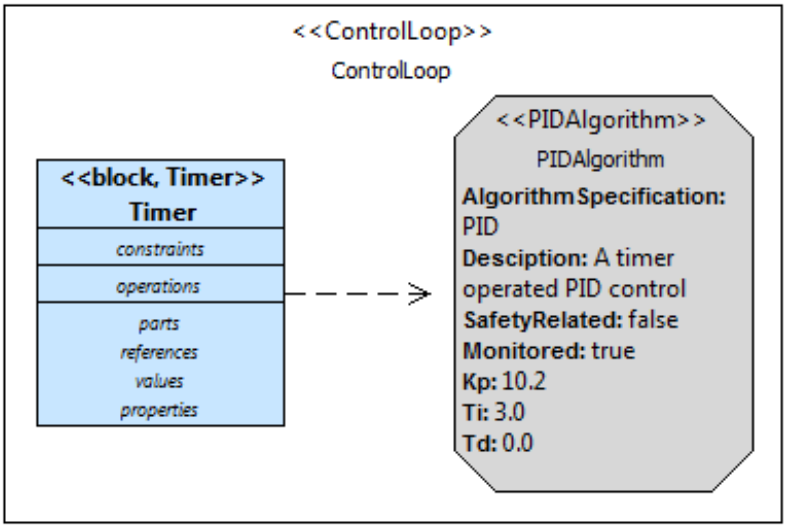
Kuva 4.18. *DevicesAndResources-aliprofiilin yleisnäkymä [31]*

Resources

Resurssien mallinnus on oleellista erityisesti hajautettujen järjestelmien yhteydessä, joissa ainakin osa resursseista voi olla useiden hajautettujen ohjelmistokomponenttien käytössä. Tällaisissa tilanteissa pitää resursseja suojata eri ohjelmistokomponenttien yhtäaikaiselta käytöltä, tai ainakin yhtäaikaisen käytön vaikutukset tulee ottaa huomioon. [28]

DevicesAndResources-aliprofiilin Resources-pakkaus sisältää, nimensä mukaisesti, erilaisten resurssien mallintamiseen liittyvää käsitteistöä. Pakkauksen ylimmän tason käsite on resurssi (Resource), joka jakaantuu neljää pääaliluokkaan, jotka puolestaan ovat passiivinen, aktiivinen, fyysinen ja looginen resurssi. Passiiviset resurssit (PassiveResource) edustavat resursseja, jotka vaativat ulkoisen herätteen. Tällaisia ovat muun muassa D/A-muuntimet, joilta muunnos pitää erikseen pyytää. Aktiiviset resurssit (ActiveResource) sitä vastoin edustavat resursseja, jotka pystyvät itse generoimaan herätteitä ja signaaleja. Tällaisia resursseja ovat muun muassa prosessorit (Processor) ja ajastimet (Timer), jotka generoivat herätteen, kun asetettu aika on kulunut. Resurssit jaetaan myös fyysisiin ja loogisiin. Fyysiset resurssit (PhysicalResource) edustavat jotakin käsin kosketeltavaa, eli muun muassa D/A-muunnin voi olla fyysinen resurssi. Looginen resurssi (LogicalResource) edustaa puolestaan resurssia, joka on olemassa esimerkiksi sähköisessä muodossa. Looginen resurssi voi siis olla esimerkiksi ohjelmistoprosessi tai tiedostojärjestelmä.

Kuvassa 4.19 on annettu esimerkki ajastimen soveltamisesta automaatioprofiilin kontekstissa. Esimerkissä ajastin on asetettu ajastamaan säätöalgoritmin suorittamista. Kuvan esitystä on yksinkertaistettu ja siitä puuttuu lukuisia normaaliin säätöpiiriin kuuluvia elementtejä.

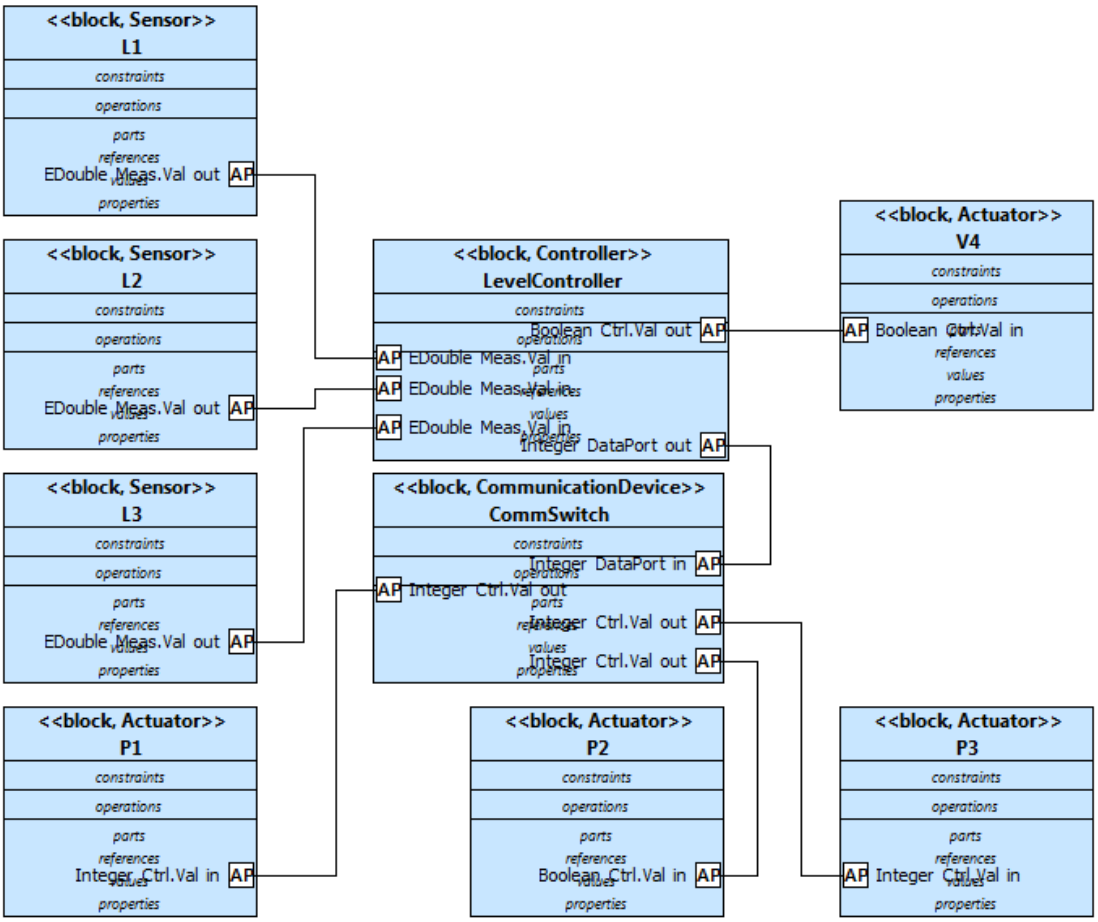


Kuva 4.19. *Esimerkki Resources-pakkauksen käsitteiden soveltamisesta [30]*

PhysicalDevices

Automaatiojärjestelmä sisältää lähes poikkeuksetta automaatiolaitteiston. Ohjelmiston kannalta minimilaitteisto on luonnollisesti ajoympäristö, joka pystyy suorittamaan ohjelmakoodia. Ajoympäristöjä voivat olla esimerkiksi tavalliset PC-tietokoneet tai mikrokontrollerit. Automaatiolaitteistoon kuuluu kuitenkin lähes poikkeuksetta myös muuta laitteistoa. Ensinnäkin tarvitaan laitteistoa, jolla voidaan vaikuttaa automatisoitavaan prosessiin, eli toimilaitteita, kuten venttiilejä, pumppuja ja moottoreita. Toisekseen tarvitaan laitteistoa, jolla voidaan mitata prosessista haluttuja suureita, eli antureita, kuten lämpötila-, paine ja virtausantureita. Lisäksi laitteistoon voidaan lukea vielä mahdolliset kenttäväylät oheislaitteineen, käyttöliittymät järjestelmään ja suoritinyksiköt, kuten säätimet ja prosessiasemat, jotka ajavat automaatiosovellusta.

PhysicalDevices-pakkaus sisältää käsitteistöä fyysisen laitteiston mallintamiseen. Pakkauksen keskeinen käsite on laite (Device), joka on kaikkien muiden pakkauksessa määriteltyjen laitteiden pohjakäsite. Pakkaus määrittelee automaatiolle tyypillisiä laitetyyppejä kuten toimilaite (Actuator), anturi (Sensor), säädin (Controller) ja kommunikointilaite (CommunicationDevice). Tärkeä pakkauksen käsite on myös järjestelmäalue (SystemArea), jonka avulla voidaan rajoittaa sekä ohjata tapahtumien ja viestin leviämistä automaatiojärjestelmässä. Kuvassa 4.20 on annettu esimerkki pakkauksen käsitteiden hyödyntämisestä. Esimerkissä on määritelty yksinkertainen säätöjärjestelmän laitemalli, johon kuuluu antureita (L1-L3) ja toimilaitteita (V4, P1-P3), säädin (LevelController) sekä kommunikointilaite (CommSwitch).



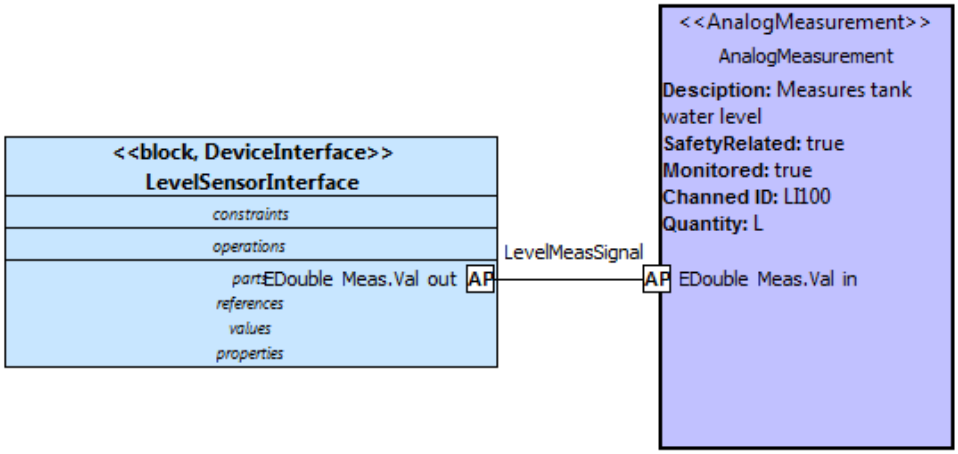
Kuva 4.20. *Esimerkki PhysicalResources-pakkauksen käsitteiden soveltamisesta (mukailtu lähteestä [30])*

DeviceInterfaces

Automaatio-ohjelmisto ei yleensä ole kykenevä suoraan kommunikoimaan erilaisten automaatiolaitteiden kanssa. Jokaisen laitteen kommunikointiprotokollien koodaaminen osaksi automaatio-ohjelmistoa olisi kuitenkin valtava urakka. Tästä syystä laitteiden ja ohjelmistojen väliin lisätään yleensä laitteita kuvaavia abstrakteja kerroksia, jotka helpottavat laitteiden käsittelyä. Näitä kerroksia kutsutaan usein ajureiksi tai laiterajapinnoiksi.

DeviceInterfaces-pakkaus sisältää edellä kuvattuja laiterajapintoja edustavia käsitteitä. Pakkauksen käsitteiden avulla voidaan ohjelmistosta viitata laitteeseen hyvin määritellyn laiterajapinnan kautta. Pakkaus määrittelee laiterajapinnat muun muassa toimilaitteelle ja anturille. Toimilaitteen laiterajapinnalla (ActuatorInterface) voidaan antaa ohjauksia toimilaitteelle ja anturin laiterajapinnalla (SensorInterface) lukea anturin mittausarvo. Lisäksi pakkaus sisältää erilaisia erikoistettuja portteja käytettäväksi laiterajapintojen yhteydessä, joilla voidaan muun muassa konfiguroida laitteen parametreja. Lisäksi laiterajapinnoille on määritelty yleisiä toimintoja, jotka mahdollistavat laiterajapintojen käytön suunnittelussa, vaikka lopullista laitetyyppiä ei vielä tiedettäisikään.

Kuvassa 4.21 on esitetty laiterajapintojen soveltamisen periaate. Laiterajapinta edustaa ohjelmistolle laitetta. Kuvan esimerkin tapauksessa mittaussisääntulo käyttää anturin rajapintaa lukeakseen mittaustuloksen.

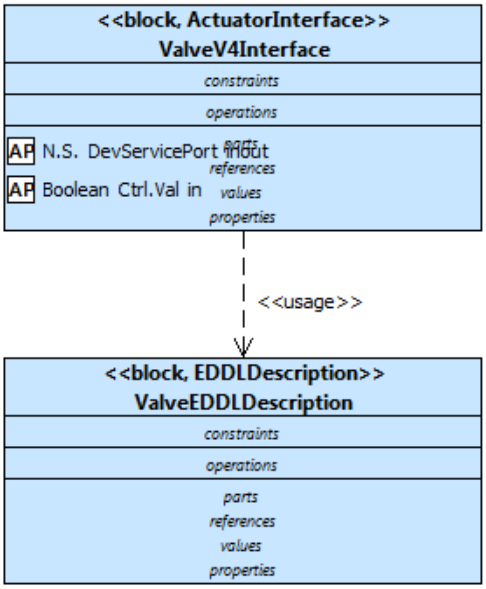


Kuva 4.21. *Esimerkki DeviceInterface-pakkauksen käsitteiden soveltamisesta (mukailtu lähteestä [30])*

FDTInterface & EDDL

DevicesAndResources-aliprofiilin kaksi viimeistä pakkausta ovat nimeltään FDTInterface ja EDDL. FDT (Field Device Tool) ja EDDL (Electronic Device Description Language) ovat laitteiston ja ohjelmiston välisiä rajapintakuvauksia standardoivia tekniikoita. Pakkaukset sisältävät käsitteistöä laiterajapintojen ja -kuvausten standardisoituun esittämisen automaatioprofiilin kontekstissa.

DTM-rajapinta (Device Tool Management) on FDT:n käyttämä standardi rajapintakuvaus, jota voidaan hyödyntää automaatioprofiilin laiterajapintojen tilalla. Kuvan 4.21 tilanteessa DTM-rajapinta (DTMInterface) voitaisiin sijoittaa suoraan anturirajapinnan (SensorInterface) tilalle. EDDL-kuvaus (EDDLDescription) puolestaan tarjoaa joustavan tavan kuvata laitteen rajapintaa ja sen mahdollistamia toimintoja. EDDL-kuvaus on siis pikemminkin standardoitu tapa esittää laitekuvaus, kuin varsinainen laiterajapinta. Kuvassa 4.22 on esitelty EDDL-kuvauksen soveltamista automaatioprofiilin kontekstissa. Esimerkissä automaatioprofiilissa määritelty toimilaiterajapinta käyttää EDDL-kuvausta laitteen rajapinnan ja mahdollisuuksien hyödyntämisessä.



Kuva 4.22. *Esimerkki EDDL-kuvauksen soveltamisesta (mukailtu lähteestä [30])*

4.4. UML AP-työkalu

Edellä esitelty UML-automaatioprofiili sisältää automaation toimialakohtaista käsitteistöä, jota voidaan hyödyntää sekä yleisenä suunnitteluosapuolten välisenä käsitteistönä, että UML-mallinnuskäsitteistönä. Koska automaatioprofiili ei ole vielä laajassa käytössä, ei sitä tukevia työkaluja ole tarjolla. Tästä syystä AUKOTON-projektissa on kehitetty UML AP-työkalu. UML AP-työkalu on alun perin Timo Vepsäläisen kehittämä UML-mallinnustyökalu, joka tukee myös automaatioprofiilin käsitteistöä. Työkalua ja sen taustalla olevia tekniikoita on tarkemmin esitelty Vepsäläisen diplomityössä: ”UML-profiilityökalu automaatiosuunnitteluun” [14] ja artikkelissa: ”Tool Support for the UML Automation Profile - for Domain-Specific Software Development in Manufacturing” [32]. Tätä kirjoitettaessa työkalu on versiossa 0.9.2. Tässä työkaluun ja sen ominaisuuksiin tutustutaan automaatiosovellusten mallipohjaisen kehitysprosessin kannalta.

AP-työkalu on kehitetty Eclipse-alustalle. Eclipse on alun perin IBM:n kehittämä, mutta vuonna 2001 avoimen lähdekoodin yhteisölle luovutettu ohjelmointiympäristö, jonka jälkeen se on ollut ilmaiseksi saatavilla. IBM-tausta antaakin Eclipselle suuren edun muihin vastaavantyyppisiin tuotteisiin nähden. Eclipse ei kuitenkaan ole pelkkä ohjelmointiympäristö vaan, kuten kappaleen alussa jo todettiin, universaali työkaluympäristö. [33]. Se on laajennettavissa niin kutsuttujen liitännäisten (plugin) avulla, joita kuka tahansa voi kehittää Eclipseen. Tällä tavalla kokonaisia ohjelmistoja voidaan toteuttaa Eclipsen päällä ajettaviksi. Myös UML AP-työkalu on tällainen liitännäinen. Eclipsen laajennettavuus ja alustaluonne mahdollistavat paljon myös tämän diplomityön varsinaista aihetta, eli kehitysprosessin ja automaatioprofiilin ohjeistusta, ajatellen. Eclipseen on muun muassa mahdollista integroida ohjeistusta sekä interaktiivisia avusteita, jotka ovat käytännössä Eclipsen liitännäisiä.

AP-työkalun tarkoitus on toimia ohjelmistona, jolla automaatio-ohjelmisto voidaan mallintaa, joten se on kehitysprosessin kannalta kantava työkalu. AP-työkalu toimii osana automaation kokonaissuunnitteluprosessia kattaen ohjelmiston mallintamisen ja suunnittelun. AP-työkalu tukee mallipohjaista kehitystä mahdollistaen kaikki mallipohjaisen kehityksen mallinnusvaiheet vaatimusmäärittelystä alustariippuvaan suunnitteluun. Tärkeää on lisäksi se, että työkaluun pystytään lisäämään liitännäisiä, jotka suorittavat mallitransformaatioita, joita mallinnusvaiheiden välillä on. Työkalu pystyy siis esimerkiksi muuntamaan vaatimusmallin alustariippumattomaksi malliksi sopivan liitännäisen avulla.

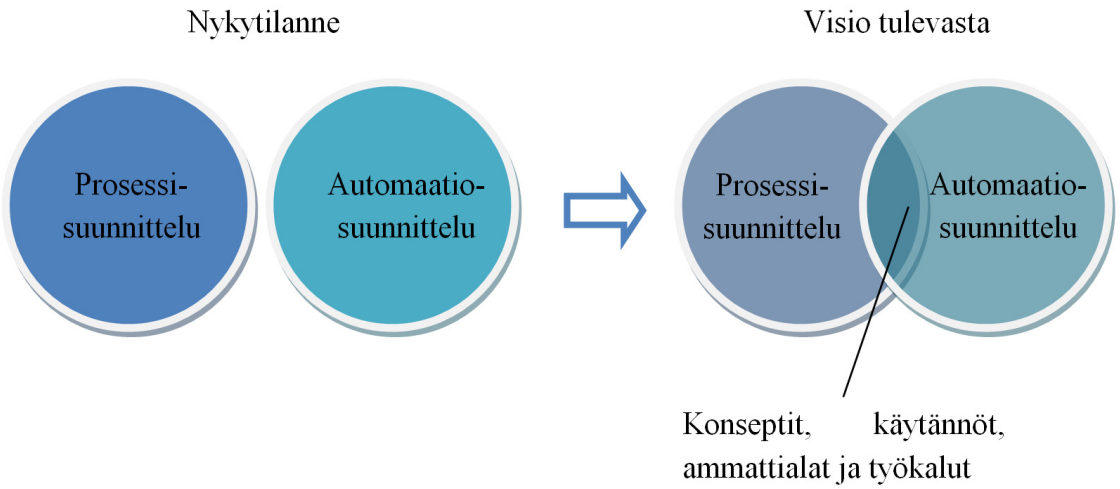
AP-työkalu integroituu sitä edeltäviin ja seuraaviin suunnittelutyökaluihin mahdollistamalla suunnittelutiedon tuonnin (import) ja viennin (export) niihin työkaluihin, joihin AP-työkalun halutaan integroituvan. AP-työkalu voi siihen lisättyjen liitännäisten avulla, integroitua esimerkiksi Intergraph ja Multiprog [34] työkaluihin. Tällöin Intergraph-suunnittelutyökalulla tuotetaan esimerkiksi tietoa prosessista, joka tuodaan AP-työkaluun, jossa tuodun tiedon perusteella mallinnetaan automaation sovellusohjelmisto. Lopuksi suunnittelutieto viedään AP-työkalusta Multiprog-työkaluun, jossa sitä käytetään lopullisen sovelluskoodin muodostamiseen.

5. AUTOMAATIOSOVELLUKSEN UML-MALLIPOHJAINEN KEHITYSPROSESSI

Tässä luvussa kuvataan AUKOTON-projektissa kehitetty automaatio-ohjelmistojen mallipohjainen kehitysprosessi. Kehitysprosessi hyödyntää luvussa neljä esitettyä mallipohjaisten automaatio-ohjelmistojen kehitysympäristöä. Ennen varsinaista kehitysprosessin kuvausta käydään läpi ne tavoitteet, joihin AUKOTON-lähestymistavalla on pyritty ja esitellään esimerkkitapaus, jonka avulla kehitysprosessin eri vaiheita kuvataan ja havainnollistetaan. Lopuksi tehdään katsaus esitetyn kehitysprosessin tarjoamiin etuihin perinteiseen suunnitteluprosessiin verrattuna.

5.1. AUKOTON-lähestymistapa automaatiosuunnitteluun

AUKOTON-projektin taustalla on ajatus automaation ohjelmistokehitysprosessin paremmasta integroitumisesta koko automaatiojärjestelmän suunnittelu- ja kehitysprosessiin. Nykykäytännön ongelmana on, kuten luvussa kaksi tuli esille, suunnitteluosapuolten hajanaisuus siten, että automaatio- ja prosessisuunnittelu ovat erillisiä kokonaisuuksia vailla yhtenäisyyttä. Kuva 5.1 havainnollistaa nykyistä ja tavoiteltua asetelmaa suunnitteluosapuolten välillä.



Kuva 5.1. *Nykyinen suunnittelukäytäntö ja AUKOTON-projektin visio tulevaisuudesta (mukailtu lähteestä [35])*

AUKOTON-projektin tavoitteena on tuoda kokonaissuunnitteluprosessiin yhtenäisiä käsitteitä, systematisoida metodeja ja tarjota työkalutukea [36]. Suurin keskittymisen

alue on nimenomaan automaatio- ja prosessisuunnittelun parempi yhteistoiminta ja tavoitteena onkin, että suunnittelualat voisivat projektissa edetä rinnakkain, ei peräkkäin [35]. Mainitut seikat ovat yleisiä, koko suunnitteluprosessin kattavia tavoitteita. AUKOTON-kehitysprosessi on kuitenkin pohjimmiltaan tarkoitettu ohjelmistokehitysprosessiksi ja sitä on tarkoitus käyttää ohjelmistopainotteisten automaatiosovellusten suunnittelussa ja toteutuksessa. Kehitysprosessilla on pyritty paremmin integroimaan ohjelmistokehitys osaksi kokonaissuunnitteluketjua ja muut tavoitteet sekä hyötynäkökohdat ovat seurausta tästä.

AUKOTON-projektissa kehitetty mallipohjainen automaatio-ohjelmistojen kehitysprosessi pyrkii siis parantamaan automaatiosovellusten ohjelmistokehitysprosessia. Tommilan et al. mukaan tavoitteena on parantaa eri suunnitteluosapuolten välistä kommunikaatiota, nostaa järjestelmän laatua panostamalla vaatimusten määrittelyyn ja konseptuaaliseen suunnitteluun, tukea elinkaaren hallintaa ja jäljitettävyyttä sekä nostaa tuottavuutta välttämällä manuaalista tiedon siirtämistä, tukemalla uudelleenkäytettävyyttä ja automatisoimalla rutiinitehtäviä [35]. Luvussa kolme kuvatut mallipohjaiset lähestymistavat luovat edellytyksiä näiden tavoitteiden saavuttamiselle. Tästä syystä uuden kehitysprosessin pohjaksi on valittu nimenomaan mallipohjainen lähestymistapa.

UML-mallipohjainen kehitysprosessi vie automaation sovelluskehityksen lähemmäksi tavanomaisen ohjelmistokehityksen mallipohjaista lähestymistapaa kuin automaatiosuunnittelussa on perinteisesti käytetty. Kehitysprosessi pohjautuu OMG:n MDA-kehitysprosessiin ja tukeutuu toimialakohtaiseen käsitteistöön, joka konkretisoituu UML-automaatioprofiilissa. Kehitysprosessi esittelee perinteiseen lähestymistapaan nähden uuden vaiheen, jossa järjestelmä mallinnetaan alustariippumattomasti. Tämä vaihe luo mahdollisuuksia kehitettyjen ratkaisuiden uudelleenkäytölle, koska ratkaisu ei ole sidottu mihinkään tiettyyn laite- ja varusohjelmistoalustaan, vaan sitä voidaan hyödyntää mille tahansa alustalle.

Vaikka pyrkimyksenä on automatisoida osia sovelluskehitysprosessista ja vähentää manuaalisen työn määrää, tarkoitus ei ole syrjäyttää automaatiosuunnittelijoita. Luovalle suunnittelutyölle jää AUKOTON-lähestymistavassakin tilansa, jota ei edes pidä pyrkiä kokonaan sivuuttamaan. Enemmänkin tarkoituksena on tuoda luova suunnittelutyö enemmän esille ja välttää manuaalista tietojen siirtelyä sekä muuta tuottamatonta ja suunnittelua edistämätöntä työtä. Lisäksi AUKOTON-kehitysprosessin työkaluilla ja käsitteistöllä voidaan toteuttaa myös perinteistä suunnittelua ja automaatiosovelluksen mallintamista ilman mallipohjaisen suunnitteluprosessin ominaisuuksia. Esimerkiksi automaatioprofiilin automaatiotoimintoja voidaan käyttää automaatiosovelluksen toiminnallisuuden suunnittelussa, vaikka tarkoitus ei olisikaan hyödyntää tätä mallia sellaisenaan toteutuksen generoinnissa kohdealustalle.

5.2. Esimerkkiprosessi

Tässä aliluvussa esitellään esimerkkiprosessijärjestelmä, jota käytetään myöhemmin tässä luvussa esiteltävän automaatiosovelluksen mallipohjaisen ohjelmistokehitysprosessin havainnollistamiseen. Esimerkki on peräisin Tommilan et al. AUKOTON-projektia ja siinä kehitettävää automaatio-ohjelmistojen mallipohjaista kehitysprosessia yleisesti käsittelevästä dokumentista: ”Seamless development path for automation applications – Concepts and approach, an overview” [35], josta se on tähän suomennettu. Prosessijärjestelmä koostuu vesitankista ja venttiilistä V4 ja sen tarkoitus on pitää riittävä määrä vettä järjestelmässä, joka on esitetty kuvassa 5.2.

Esimerkkejä prosessiin liittyvistä vaatimuksista:

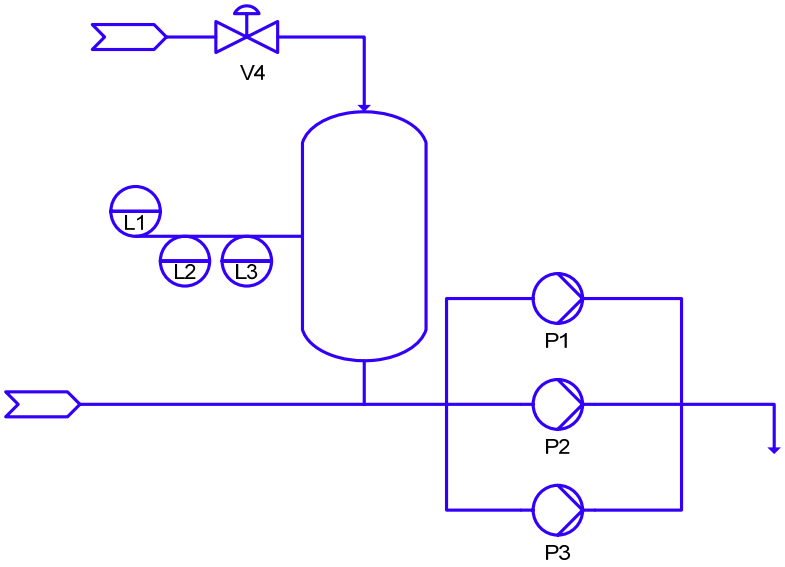
- Tankissa tulee aina olla riittävästi vettä, jotta vuodoista ja veden kulutuksessa tapahtuvista vaihteluista selvitään veden loppumatta.
- Jos tankki tyhjenee, pumput P1…P3 tulee sammuttaa, jotta vältytään pumppujen vaurioitumiselta.

Prosessiselostuksen perusteella voidaan löytää seuraavat säätötehtävät ja -strategiat:

- Säilytä veden määrä: Mittaa pinnankorkeus (seurattu prosessimuuttuja) ja säädä venttiilin V4 asentoa (säädetty prosessimuuttuja), jotta pinnankorkeus säilyy asetusarvossaan.
    - Sallittu muutos pinnakorkeudessa on +-10%.
- Suojaa pumppuja: Jos tankin pinnankorkeus laskee alle 5% pysäytä pumput 20 sekunnin kuluessa.
    - Suojauksen tulee olla luotettava.

Automaatioaste:

Normaalisti tankin toiminnot ovat täysin automaattisia, mutta operaattoria tulee informoida häiriön sattuessa.



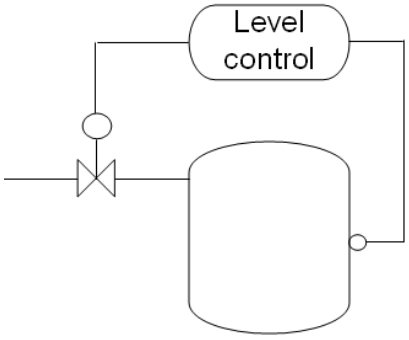
Kuva 5.2. *Esimerkkiprosessin jäähdytysvesijärjestelmä (mukailtu lähteestä [37 s. 63])*

Kun kannanotto automaatioasteesta yhdistetään prosessikuvaukseen, seuraavat vaatimukset voidaan yhdistää uuteen säätöjärjestelmään:

- Veden pinnankorkeus tankissa on mitattava 2% tarkkuudella.
- Alhainen pinnankorkeus on havaittava 99.99% varmuudella.
- Säätöventtiilin V4 asennot ovat auki/kiinni (tässä tehty suunnittelupäätös!).
- Tankin pinnankorkeutta on säädettävä, maksimi hystereesi 5%.
- Hälytys on annettava jos pinnankorkeus on liian alhaalla.

Järjestelmän toiminnallisuudet ovat:

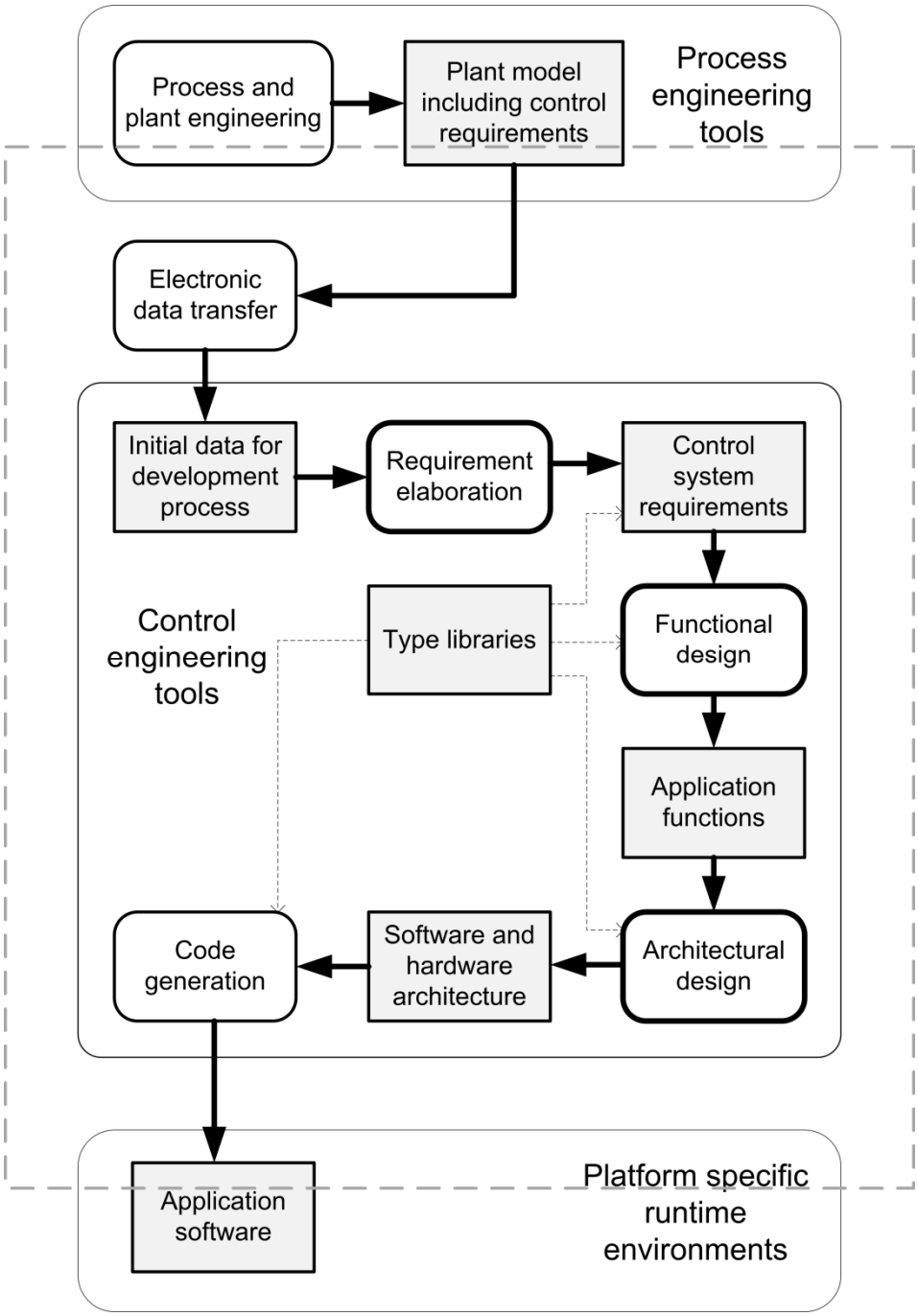
- Analogiasisäänmeno: LI-1 (pinnankorkeuden mittaus)
- Binäärisisäänmeno: LI-2, LI-3 (redundanttiset pintakytkimet)
- Binäärinen ulostulo: V4 (auki/kiinni venttiili)
- Säätötoiminnot:
    - Tankin pinnankorkeuden säätö (Kuva 5.3)
    - Suojaussignaalin generointi.



Kuva 5.3. *Yksinkertainen automaatiojärjestelmä joka koostuu mittaussisäänmenosta, säätötoiminnosta ja ohjaussignaalista [35]*

5.3. Kehitysprosessin vaiheet

AUKOTON-kehitysprosessi myötäilee luvussa kolme esitettyä mallipohjaista ohjelmistokehitysprosessia. Erityisesti esitettävä kehitysprosessi on lähellä MDA:n lähestymistapaa. Kehitysprosessi kuvataan seuraavissa aliluvuissa suunnittelijalähtöisesti siten, että pääpaino on suunnittelijan tehtävissä, mahdollisuuksissa ja rajoitteissa kehitysprosessin eri vaiheissa. Tekniset yksityiskohdat eivät ole tämän diplomityön kannalta oleellisia. Esimerkiksi transformaatioiden toiminta ei ole suunnittelijan kannalta oleellinen seikka, joten ne jätetään tässä kehitysprosessin kuvauksessa vähälle huomiolle. Kuvaus pyritään pitämään suoraviivaisena välttäen turhaa idealisointia ja toisaalta nykyiseen AP-työkaluun sidottuna, jolle diplomityössä toteutettu ohjeistus on kohdistettu. Kuvassa 5.4 esitetään kehitysprosessin päävaiheet.



Kuva 5.4. *AUKOTON-kehityspolun päävaiheet (mukailtu lähteestä [9])*

AUKOTON-kehityspolun varsinainen ohjelmistokehitysprosessi (Kuva 5.4 Control engineering tools osio) koostuu kolmesta päävaiheesta joiden välillä on automaattinen tai käyttäjäavusteinen siirtymä eli transformaatio mallista toiseen. Kehitysprosessin päävaiheet ovat vaatimusmallinnus, alustariippumattoman toiminnallisuuden määrittely ja alustariippuva suunnittelu. Edellä esitetyt konsepti ja käsitteet tukevat näin omalta osaltaan kehitysprosessia. Tyyppikirjastoihin voidaan tallentaa ratkaisuita, joita voidaan käyttää uudelleen myöhemmissä projekteissa.

AUKOTON-kehitysprosessin vaiheet voidaan karkeasti rinnastaa perinteisen automaatiosuunnittelun vaiheisiin, joita esiteltiin luvussa kaksi. Vaatimusmallinnus vastaa karkeasti esisuunnittelua. Esisuunnitteluvaiheessahan koottiin järjestelmän vaatimukset, mikä on myös vaatimusmallinnusvaiheen tarkoitus AUKOTON-kehitysprosessissa. Periaatteessa vaatimusmallinnusvaiheessa jo asiakas voisi mallintaa vaatimukset suoraan osaksi mallia.

AUKOTON-kehitysprosessin alustariippumattomassa suunnitteluvaiheessa suunnitellaan ja mallinnetaan järjestelmän automaatiokonsepti, eli määritellään millaisilla toiminnoilla vaatimuksissa määritellyt järjestelmän ominaisuudet saadaan toteutettua. Alustariippumaton suunnitteluvaihe vastaakin näin ollen osittain perinteisen suunnitteluvaiheistuksen perussuunnitteluvaihetta, mutta kyseessä on kuitenkin eri asia. Alustariippumattomassa vaiheessa mallinnettava automaation toiminnallisuus on korkean abstraktiotason mallinnuskielellä kuvattu, joten asiakas pystyy tulkitsemaan sitä helpohkosti toimittajaa valitessaan.

AUKOTON-kehitysprosessi viimeinen varsinainen suunnitteluvaihe, eli alustariippuva suunnittelu, sisältää samoja toimenpiteitä kuin perinteisen lähestymistavan toteutussuunnitteluvaihe. Toteutussuunnitteluvaiheessa järjestelmä suunnitellaan niin tarkasti, että sen perusteella voidaan tehdä järjestelmän toteutus. AUKOTON-kehitysprosessin alustariippuvassa vaiheessa järjestelmä mallinnetaan samalla tarkkuustasolla kuin perinteisen lähestymistavan toteutussuunnitteluvaiheessa. Erona on lähinnä se, että AUKOTON-kehitysprosessissa toteutus saadaan mallista lähes suoraan generoimalla, kun taas perinteisessä lähestymistavassa toteutus tehdään yleensä erikseen suunnittelun tulosten pohjalta. AUKOTON-kehitysprosessin vaiheet kuvataan tarkemmin seuraavissa aliluvuissa.

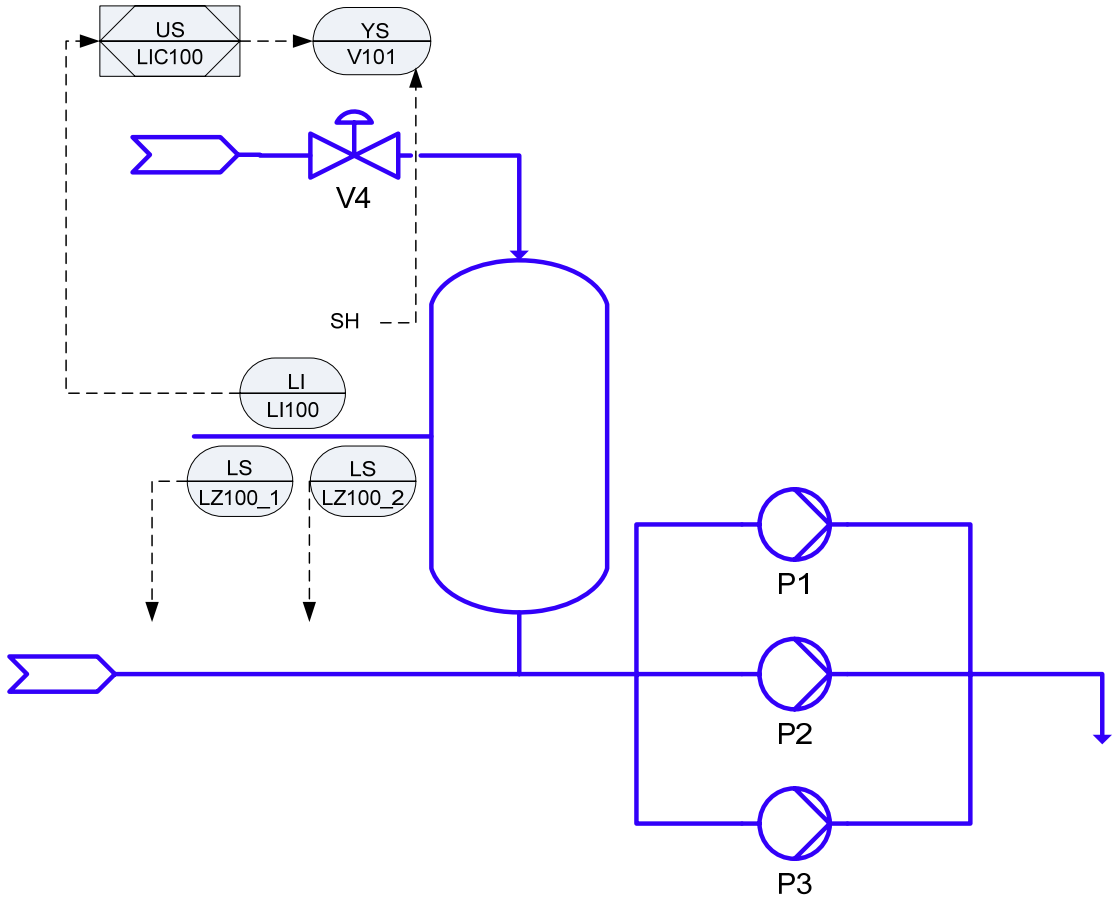
5.3.1. Edeltävät suunnitteluvaiheet

Kuten luvussa kaksi esitettiin, ennen varsinaista sovelluskehitysvaihetta, jota AUKOTON-kehitysprosessi siis edustaa, on laitos- tai automaatioprojektissa tehty huomattava määrä suunnittelutyötä. Koska ohjelmistosuunnittelun rooli on yleensä toteuttaa toiminnallisuus, jonka edeltävä suunnittelu automaatiojärjestelmälle määrittää, ei ohjelmistokehitysvaiheessa yleensä enää oteta kantaa koko automaatiojärjestelmän määrittelyseikkoihin. Tästä syystä myös AUKOTON-kehitysprosessi hyödyntää edeltäneissä suunnitteluvaiheissa tuotettua suunnittelutietoa. Ero AUKOTON-kehitysprosessin ja perinteisen kehitysprosessin välillä tulee esiin siinä tavassa, jolla tieto prosessi- ja automaatiosuunnittelijoiden välillä kulkee.

AUKOTON-kehitysprosessi pyrkii mahdollisimman saumattomaan tiedonsiirtoon, jossa manuaaliset työvaiheet minimoituvat. Vaikka myöhemmin tullaan käsittelemään vain kahta mahdollista lähtötietoformaattia, on syytä korostaa, että AUKOTON-kehitysprosessi ei rajoita lähtötietojen muotoa. Mikä tahansa sopiva tietoformaatti, jolle on tarvittava vastine kehitysprosessin työkalussa, voi toimia lähtötietojen siirtämisen välineenä.

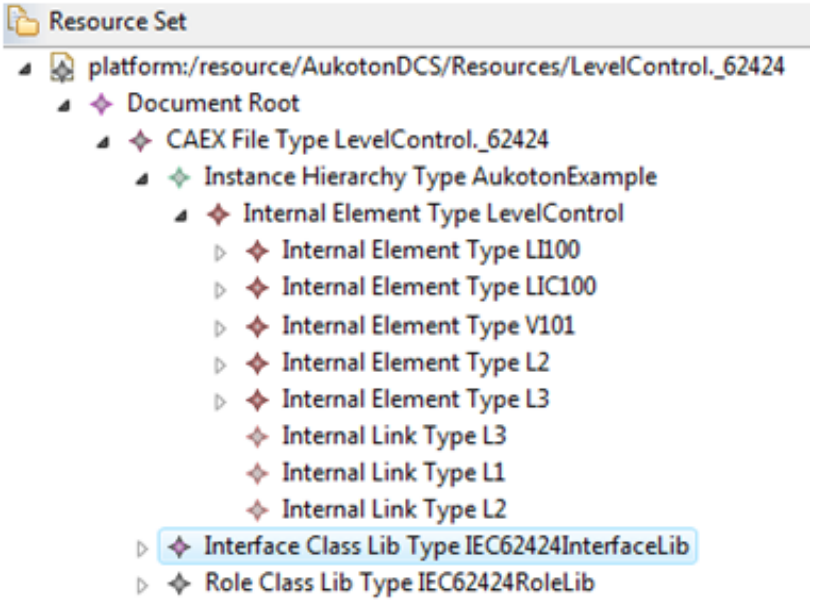
Ohjelmistosuunnittelun kannalta oleellisimpia lähtötietoja automaatio- ja prosessisuunnittelusta ovat ohjaus- ja mittauskohteiden väliset riippuvuudet sekä ohjaus- mittauskohteiden vaatimukset, joita voivat olla muun muassa hälytysrajat [9]. Mainitut tiedot ovat oleellisia ohjelmiston kannalta, koska ne määrittelevät sen automaatiojärjestelmän toiminnallisuuden, joka ohjelmistolla halutaan toteuttaa. Koska AUKOTON-kehitysprosessin pyrkimyksenä on saumaton tiedonsiirto, tulee nämä tiedot esittää mahdollisimman helposti koneellisesti käsiteltävässä muodossa. AUKOTON-prosessissa voidaankin hyödyntää esimerkiksi standardia IEC 62424 [38], joka määrittelee tiedonsiirtoa prosessisuunnittelusta automaatiosuunniteluun [9]. IEC 62424 määrittelee, miten automaation kannalta oleellista suunnittelutietoa esitetään PI-kaavioita täydentävinä lisämääreinä. Nämä lisämääreet voidaan puolestaan esittää CAEX-formaatissa (Computer Aided Engineering Exchange), jossa lisämäärein esitetty tieto on tallennettu rakenteisessa XML-muodossa.

AUKOTON-kehitysprosessi ei edellytä IEC 62424:n käyttöä lähtötietojen kuvaamiseen vaan se on vain yksi mahdollisista vaihtoehdoista. Koska IEC 62424 on tätä kirjoitettaessa toinen kehitysprosessissa käytetyistä lähtötietoformaateista, käytetään sitä tässä esimerkinomaisesti kuvaamaan ohjaus- ja mittapistesuunnittelun tuloksia. Kuvassa 5.5 on esitetty esimerkkivesiprosessin PI-kaavio, johon on lisätty IEC 62424:n mukaisia lisämääreitä. Kuvassa 5.5 voidaan nähdä muun muassa lisämääre LIC100, joka säätää tankin pinnankorkeutta. LIC100 ohjaa venttiiliä V4 venttiilinohjaimen V101 kautta. Venttiilinohjain käyttää signaalia SH lukitsemaan venttiilin, jos tankissa on liikaa nestettä. Kuvassa 5.5 määritellään myös pintakytkimet LZ_100_1 ja LZ_100_2. Nämä kytkimet aktivoituvat, jos veden pinta laskee alle määritellyn turvarajan ja aktivoivat lukitussignaalit, jolloin pumput P1-P3 tulee sammuttaa.



Kuva 5.5 *IEC 62424 lisämäärein tarkennettu vesiprosessin PI-kaavio (mukailtu lähteestä [39])*

Kuvassa 5.5 esitetyt PI-kaavion lisämääreet asettavat automaatio- tai prosessisuunnittelijat. Lisämääreiden perusteella generoidaan automaation vaatimuksia kuvaava CAEX-tiedosto, jossa lisämäärein esitetty tieto on esitetty tietokoneelle sopivassa rakenteellisessa XML-muodossa. CAEX-tiedosto sisältää siis muun muassa informaatiota siitä, mistä mittauksista ohjaukset riippuvat ja mitä signaaleja anturit sekä toimilaitteet tuottavat ja tarvitsevat toimintansa suorittamiseen. Kuvassa 5.6 on esitetty esimerkkiprosessista generoitu CAEX-tiedosto, joka sisältää prosessin automaatiojärjestelmälle kohdistamat vaatimukset. CAEX-rakenteessa voidaan nähdä samoja elementtejä, joita PI-kaavioon määriteltiin IEC 62424:n lisämääreillä.



Kuva 5.6. *IEC 62424 lisämääreiden perusteella generoidun CAEX-tiedoston rakennetta. Näkymä UML AP-työkalusta.*

IEC 62424 ja CAEX sopivat hyvin siihen tarkoitukseen, johon ne on alun perin tarkoitettu, eli suunnittelutiedon siirtämiseen prosessisuunnittelusta automaatiosuunnitteluun. Näillä muodoilla ei kuitenkaan ole tarkoitus siirtää kaikkea sitä tietoa, jonka edeltävät suunnitteluvaiheet tuottavat ohjelmiston kehitystä varten. AUKOTON-prosessissa hyödynnetäänkin IEC-standardien lisäksi Excel-taulukoita, joilla voidaan siirtää hyvin erilaisia lähtötietoja. Taulukoilla voidaan esittää ohjelmistosuunnittelua edeltävien vaiheiden tuloksia, kuten IO-, instrumentointi-, lukitus- ja laitelistoja. Lisäksi tyypillistä on erilaisten raja-arvojen ja viestien esittäminen Excel-taulukoiden avulla. Excel-taulukoiden rakenne ei ole standardoitu.

Kuvassa 5.7 on esitetty esimerkinomainen Excel-lähtötietotaulukko, jossa on listattu esimerkkiprosessin mittapisteet. Taulukossa on määritelty signaaleille muun muassa raja-arvoja kuten minimi- ja maksimiarvot, haluttu tarkkuus, sekä hälytysrajat. Excel-taulukoissa esitetty tieto voi olla hyödyllistä joko vaatimusmallin luomisessa, esimerkiksi lukituslistan tapauksessa tai sitten tietoa voidaan käyttää myöhemmin kehitysprosessin aikana esimerkiksi laitemallin generointiin, jos taulukossa on esitettynä järjestelmän laitteisto tai osa siitä.

Version	Process system	Process item type	Process function type	ID	Description	IO type	IO connection type	Instrument type	Range, min.	Range, max.	Engineering unit	Accuracy, %	Filtering time, s	Lower low limit	Low limit	High limit	Higher high limit	Process medium
0.1	CWS.T1	tank	level	LI100	Cooling water level	AI	4-20 mA	level transmitter	0.0	2.5	m	2	3	1.0	1.2	1.8	2.0	Water
0.1	CWS.T1	tank	level	LS100_1	Cooling water level low	BI	24 V	level switch										Water
0.1	CWS.T1	tank	level	LS100_2	Cooling water level low	BI	24 V	level switch										Water
0.1	CWS.V4	block valve	position	V101-S1	Make-up water valve closed	BI	24 V	limit switch										Water
0.1	CWS.V4	block valve	position	V101-S2	Make-up water valve open	BI	24 V	limit switch										Water
0.1	CWS.V4	block valve	position	V101-Q1	Make-up water solenoid valve	BO	24 V	solenoid valve										Water

Kuva 5.7. *Esimerkkiprosessin mittapisteet listattuna Excel-lähtötietotaulukossa [39]*

AUKOTON-kehitysprosessissa automaatiojärjestelmän suunnittelu lähtee liikkeelle vastaavasti kuin perinteinen automaatioprojekti. Suurin ero prosesseissa tulee esiin siinä, miten suunnittelutieto esitetään ja kuinka se saadaan siirrettyä ohjelmistokehitysprosessiin. Standardeja tiedonsiirtotapoja hyödyntäen tiedonsiirto voidaan toteuttaa mahdollisimman automaattisesti siten, että tieto saadaan siirrettyä edeltävistä suunnitteluvaiheista ohjelmistosuunnitteluun ilman merkittävää manuaalista työtä.

Sillä, millä työkaluilla ja tavoilla automaation ohjelmistokehityksen tarvitsemat lähtötiedot kerätään ja koostetaan, ei ole ohjelmistosuunnittelun kannalta väliä. Oleellista AUKOTON-prosessin kannalta on se seikka, että ohjelmistosuunnittelua edeltävissä vaiheissa tuotetut lähtötiedot ovat saatavilla ohjelmistokehitysvaiheeseen ja että ne ovat ilman merkittävää manuaalista työtä ohjelmistokehitystyökalussa käytettävissä. Parhaassa tapauksessa prosessi-, automaatio- ja automaation ohjelmistosuunnittelu on toteutettu käyttäen yhteensopivaa käsitteistöä, jota automaatioprofiili tukee. Tällöin yhteisymmärrys eri osapuolten välillä pystytään maksimoimaan.

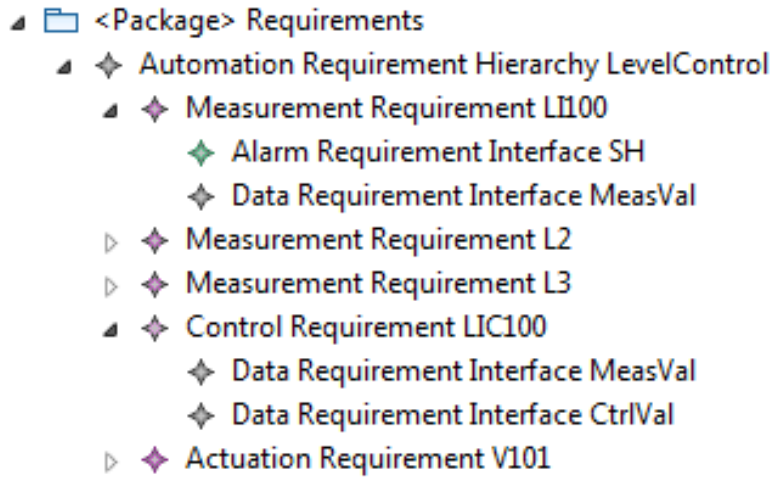
Kehitysprosessia edeltävät suunnitteluvaiheet voidaan MDA-prosessin kannalta nähdä laskentariippumattoman mallin aineiston kokoamisena. MDA ei määrittele laskentariippumattoman mallin muotoa, joten kaikkea vaatimusten esittämistä, eli mallintamista, voidaan pitää laskentariippumattoman mallin toteuttamisena.

5.3.2. Lähtötietojen tuonti AP-työkaluun

Edellisessä suunnitteluvaiheessa koottiin yhteen järjestelmän lähtötiedot, jotka koostuvat automaatiojärjestelmän vaatimuksista ja muista lähtötiedoista. Tässä vaiheessa nämä lähtötiedot tuodaan AP-työkaluun. Eräs AUKOTON-kehitysprosessin tavoitteista oli vähentää manuaalista tiedonsiirtoa eri suunnitteluvaiheiden välillä. Tästä syystä AUKOTON-ketjussa hyödynnetäänkin automaattista import-mekanismia lähtötietojen tuomiseksi työkaluun. Edellytyksenä lähtötietojen tuomiselle on sopivan lähtötietotuojan, eli import-laajennoksen (plugin), olemassaolo AP-työkalussa. Import-laajennuksen tehtävä on muuttaa lähtötiedot automaatioprofiilissa määritellyiksi elementeiksi, esimerkiksi vaatimushierarkioiksi ja vaatimuksiksi. Tätä kirjoitettaessa AP-työkaluun on integroitu lähtötietotuoja IEC 62424- ja Excel-taulukkomuotoisten lähtötietojen tuomiseen, mutta sopivalla lähtötietotuojalla työkaluun voidaan tuoda myös muussa muodossa ole-

via lähtötietoja. Tämä on tärkeä ominaisuus, koska näin luodaan edellytyksiä kehitysprosessin integroitumiseen erilaisiin laitos- ja prosessisuunnitteluympäristöihin ja -työkaluihin.

Varsinainen automaatiosovelluksen kehitysprosessi alkaa tässä vaiheessa. Lähtötiedot tuodaan AP-työkaluun sopivalla lähtötietotuojalla. Kukin lähtötietotiedosto tuodaan erikseen ja jokaiselle muodolle on oma importer-toteutuksensa AP-työkalussa. Lähtötietojen tuominen työkaluun on suoraviivainen tehtävä, eikä välttämättä edes vaadi suunnittelijalta erityistoimenpiteitä. Kuvassa 5.8 on esitetty osa esimerkkiprosessin lähtötiedoista generoitunutta vaatimusmallia, kun IEC 62424- ja Excel-lähtötiedot on tuotu työkaluun.



Kuva 5.8. *Näkymä sovelluksen vaatimusmallista lähtötietojen tuomisen jälkeen*

Lähtötiedoista generoitu vaatimusmalli pitää sisällään AutomationRequirementHierarchy-elementtejä (vaatimushierarkia), jotka järjestävät vaatimuksia eri kategorioihin. Vaatimukset ovat puolestaan AutomationRequirement-elementtejä (automaatiovaatimus), jotka ovat rakenteisia, eli tässä tietokoneen käsiteltävissä olevia vaatimuksia. Vaatimuksia tarkentavat edelleen niiden rajapinnat (Data- ja AlarmRequirementInterface). Automaatioprofiili määrittelee myös epäformaaleja vaatimustyyppejä, mutta niiden soveltuvuus automaattisesti hyödynnettäviksi automaatiotoimintojen lähtötiedoiksi on heikko. Epäformaaleilla vaatimuksilla voidaan kuitenkin selkeästi, ilman automaatiovaatimusten formaalia esitystapaa, esittää yleisiä ja laajempia vaatimuksia, jotka usein koskettavat järjestelmän yleistä toimintaa. Myös näitä vaatimuksia voidaan tuoda edeltävistä suunnitteluvaiheista työkalun vaatimusmalliin.

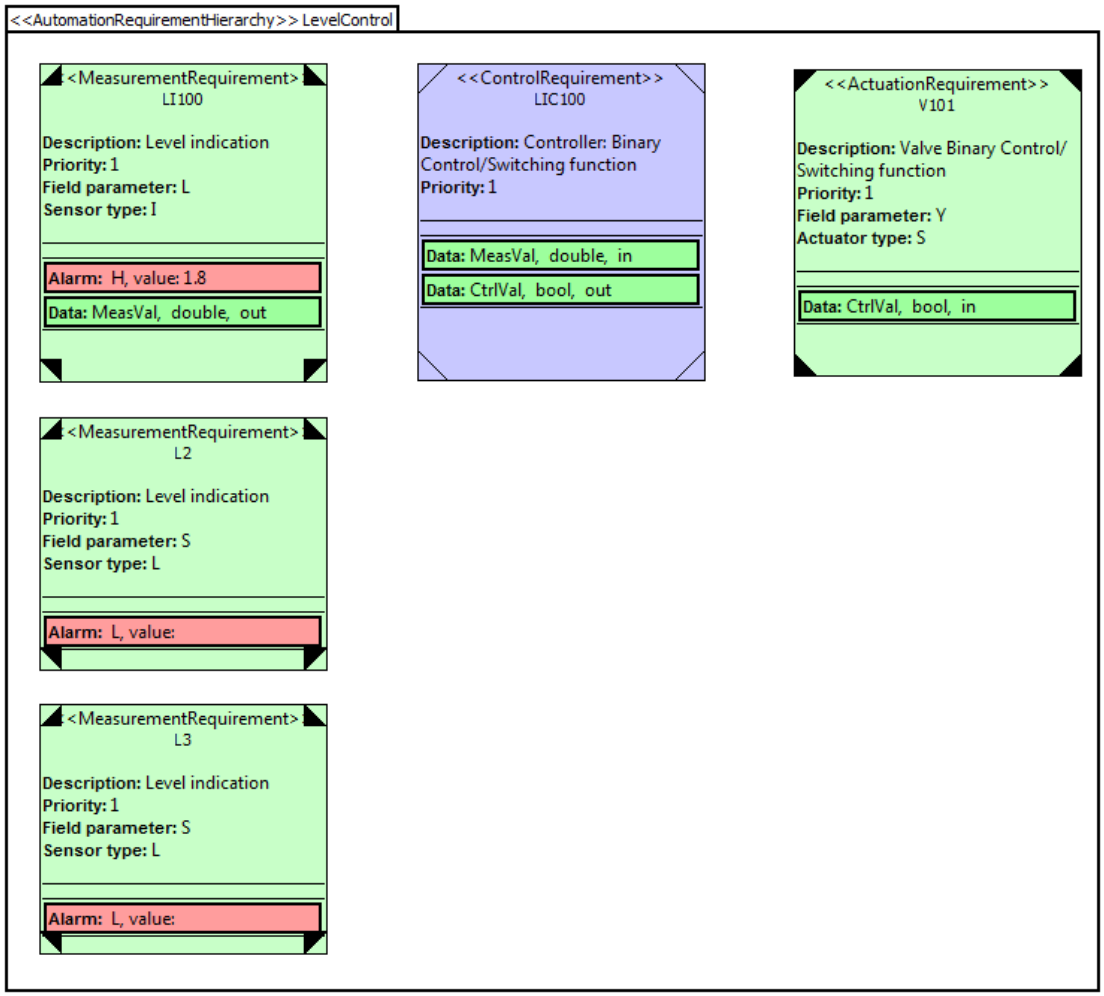
Tämä vaihe voidaan MDA-prosessin kannalta nähdä laskentariippumattoman mallin rungon luomisena. Vaatimusmallin runkoa täydennetään ja tarkennetaan seuraavassa suunnitteluvaiheessa. Erona MDA-prosessiin nähden voidaan pitää lähtötietojen sähköistä tuontia, jota MDA ei vaadi käytettäväksi, mutta ei sitä toisaalta estäkään [16].

5.3.3. Vaatimusten tarkentaminen

Lähtötietojen tuomisen jälkeen niistä generoitu vaatimusmalli on suunnittelijoiden käytettävissä. AUKOTON-prosessin eräs tavoite on painottaa suunnittelun alkuvaiheita ja siirtää painoa pois suunnittelun alustariippuvalta osalta. Näin ollen kehitysprosessissa pyritäänkin jo vaatimukset mallintamaan mahdollisimman tarkasti. [5]

Tässä suunnitteluaskeleessa vaatimusmallia tarkennetaan ja muokataan. Pääasiallisia suunnittelijan aktiviteetteja ovat vaatimuskaavion graafisen esityksen toteuttaminen ja järjestäminen, mallin tarkentaminen manuaalisesti lisäämällä tai muokkaamalla olemassa olevia vaatimuksia, vaatimusten luokittelu hierarkioihin ja näkökulmiin. Kuvassa 5.9 on graafisesti esitetty ja järjestetty esimerkkiprosessin vaatimusmalli, jolle ei ole tehty muita toimenpiteitä.

[5] Tässä on hyvä huomata, että osa siitä määrittelystä, joka on ennen tehty suoraan laitekomponenttien välillä, tehdään AUKOTON-prosessissa jo vaatimusmäärittelyssä. Tämä saattaa ensin vaikuttaa ainoastaan suunnittelutyön siirtämisellä toiseen paikkaan, mutta ideana on siirtää osa suunnittelusta sellaiseen muotoon, missä se on mahdollisimman tehokkaasti uudelleen käytettävissä. On selvää, että kaikki se, mitä ennenkin on projektissa jouduttu määrittelemään, joudutaan määrittelemään myös AUKOTON-prosessissa, koska informaatiota ei pystytä luomaan tyhjästä. Osaa työstä voidaan kuitenkin käyttää uudelleen suoraan myös tulevissa projekteissa, mikä vähentää työmäärää tällöin.

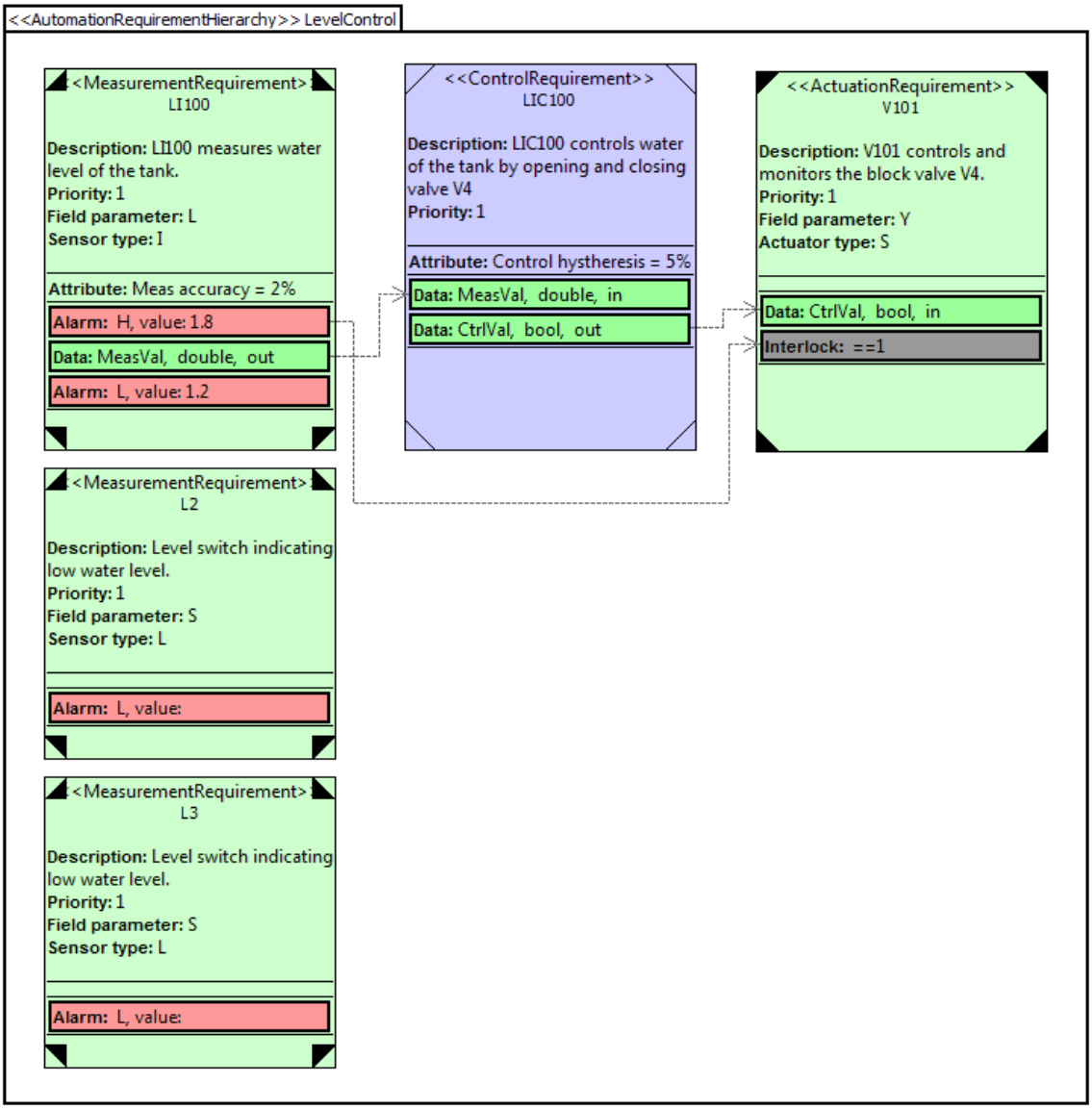


***Kuva 5.9**. Graafisesti esitetty ja järjestetty vaatimusmalli*

Vaatimukset luokitellaan koneellisesti niiden tuonnin aikana. Näin ei kuitenkaan aina päästä tyydyttävään lopputulokseen vaan luokittelua hierarkioihin voidaan joutua muokkaamaan manuaalisesti. Eräs syy luokitella vaatimuksia uudelleen on sopivien näkökulmien luonti. Vaatimusmallia voidaan suodattaa erilaisilla näkökulmilla niin, että esimerkiksi asiakas näkee yleisempiä vaatimuksia ja pääsuunnittelija näkee kaikki vaatimukset. Näin eri sidosryhmiä koskevia vaatimuksia on helpompi käsitellä vaatimusmallilla. Vaatimusmalli itsessään sisältää aina kaiken tiedon ja näkökulmalla ainoastaan tuodaan tarvittava tieto esille. Esimerkkitapauksessa luokittelua tai näkökulmia ei ole tarvittu mallin pienen koon takia.

Vaatimusten tarkentaminen, muokkaaminen ja lisääminen ovat tämän suunnittelutehtävän tärkeimmät vaiheet. Yleensä lähtötietoina saadut vaatimukset ovat ainakin selittäviltä teksteiltään epätäydellisiä ja niitä pitää täydentää. Esimerkiksi kuvassa 5.9 oleva LIC100-vaatimuksen description, eli selite, on importerin LIC100:sta saamiensa lähtötietojen perusteella generoima yksinkertainen selite, jota tulee tarkentaa, jotta vaatimus olisi helposti ymmärrettävä. Muita täydennys- ja muokkauskohteita voivat olla muun muassa vaatimusten attribuutit, rajapinnat sekä niiden väliset yhteydet, vaatimukset itse ja jopa vaatimushierarkiat. Kuvassa 5.10 on esitetty esimerkkiprosessin tarken-

nettu vaatimusmalli. Tarkennukset ovat kohdistuneet lähinnä vaatimusten selitteisiin. Vaikka selitteiden tarkentaminen ei olekaan oleellista automaattisten mallimuunnosten kannalta, selkokieliset vaatimukset ovat tärkeitä niitä tulkitsevien ihmisten kannalta. Lisäksi malliin on lisätty hälytysrajapinta (LI100:an), vaatimusrajapintojen väliset yhteydet ja muutamia attribuuttitietoja.



Kuva 5.10. *Tarkennettu vaatimusmalli*

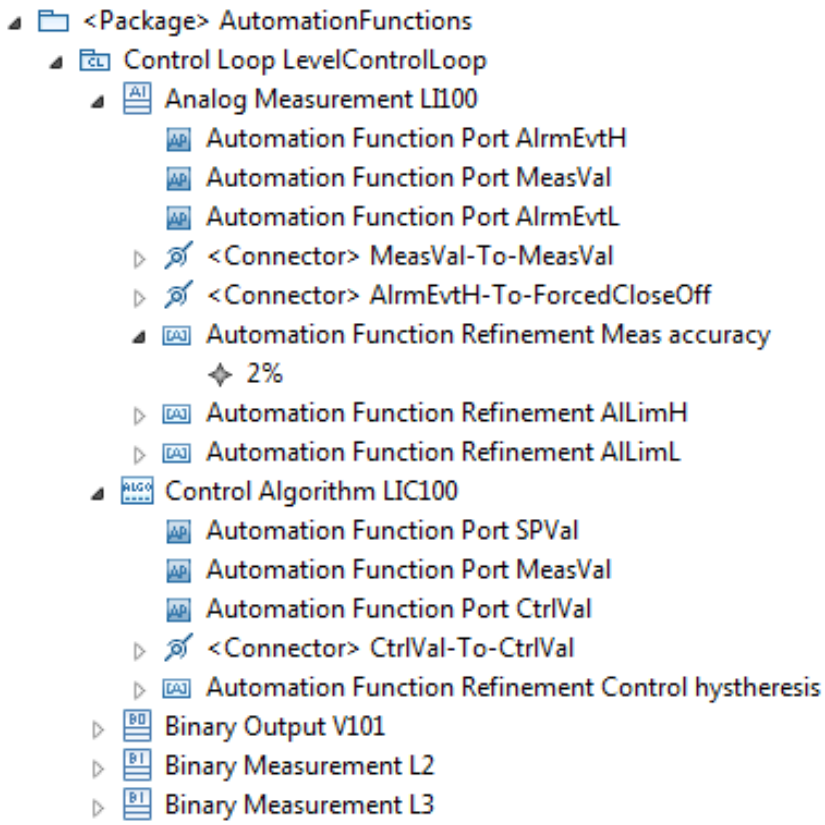
Vaatimuksia tarkennetaan, kunnes saavutetaan haluttu tarkkuustaso. Mitään selkeää rajaa vaatimusten tarkkuudelle ei ole olemassa, mutta mitä tarkemmin vaatimukset esittää, sitä vähemmän seuraavissa työvaiheissa joudutaan tekemään työtä. Vaatimusten tarkennusvaiheen lopputuloksena tulisi olla täydellinen vaatimusmalli esitettynä yhdellä tai useammalla kuvan 5.10 mukaisella vaatimuskaaviolla. Vaatimusten tarkennusvaiheessa aikaansaatu vaatimusmalli vastaa osittain MDA-kehitysprosessin laskentariippumatonta mallia, joka kuvaa järjestelmän toimintaa ilman toiminnallisia tai alustariip-

puvia toteutustekniikan yksityiskohtia. Lopullinen vaatimusmalli esittää edeltävien suunnitteluvaiheiden sovellussuunnitteluun vaikuttavat tiedot koottuna ohjelmistosuunnittelua varten.

5.3.4. Vaatimuksista automaatiotoimintoihin

Kun järjestelmän vaatimukset on koottu vaatimusmalliin, siirrytään järjestelmän mallinnuksessa alustariippumattomaan suunnitteluaskeleeseen. Alustariippumaton malli voitaisiin vaatimusten perusteella mallintaa kokonaan suunnittelijan toimesta. Tämä on kuitenkin työläs ja virhealtis tehtävä. MDA-prosessissa muunnos laskentariippumattomasta mallista alustariippumattomaan malliin suoritetaan mahdollisimman pitkälle transformaationa, joka generoi alustariippumattoman mallin tai osan siitä laskentariippumattoman mallin perusteella.

AUKOTON-kehitysprosessissa MDA-prosessia vastaava muunnos suoritetaan edellisessä vaiheessa koostetun vaatimusmallin ja alustariippumattoman mallin välillä. Mallimuunnos on pääsääntöisesti automaattinen eikä käyttäjän tässä tapauksessa tarvitse osallistua sen läpivientiin muuten, kuin valitsemalla lähtö- ja kohdepakkaukset, joiden välillä muunnos suoritetaan ja osoittamalla sopiva transformaattori suorittamaan mallimuunnos. Tämän jälkeen työkaluun integroitu transformaattori hoitaa mallimuunnoksen. Mallimuunnoksen lopputuloksena syntyy järjestelmän alustariippumaton mallin runko. Osa esimerkkijärjestelmän vaatimusmallista transformoimalla generoidusta alustariippumattomasta mallista on esitetty kuvassa 5.11.



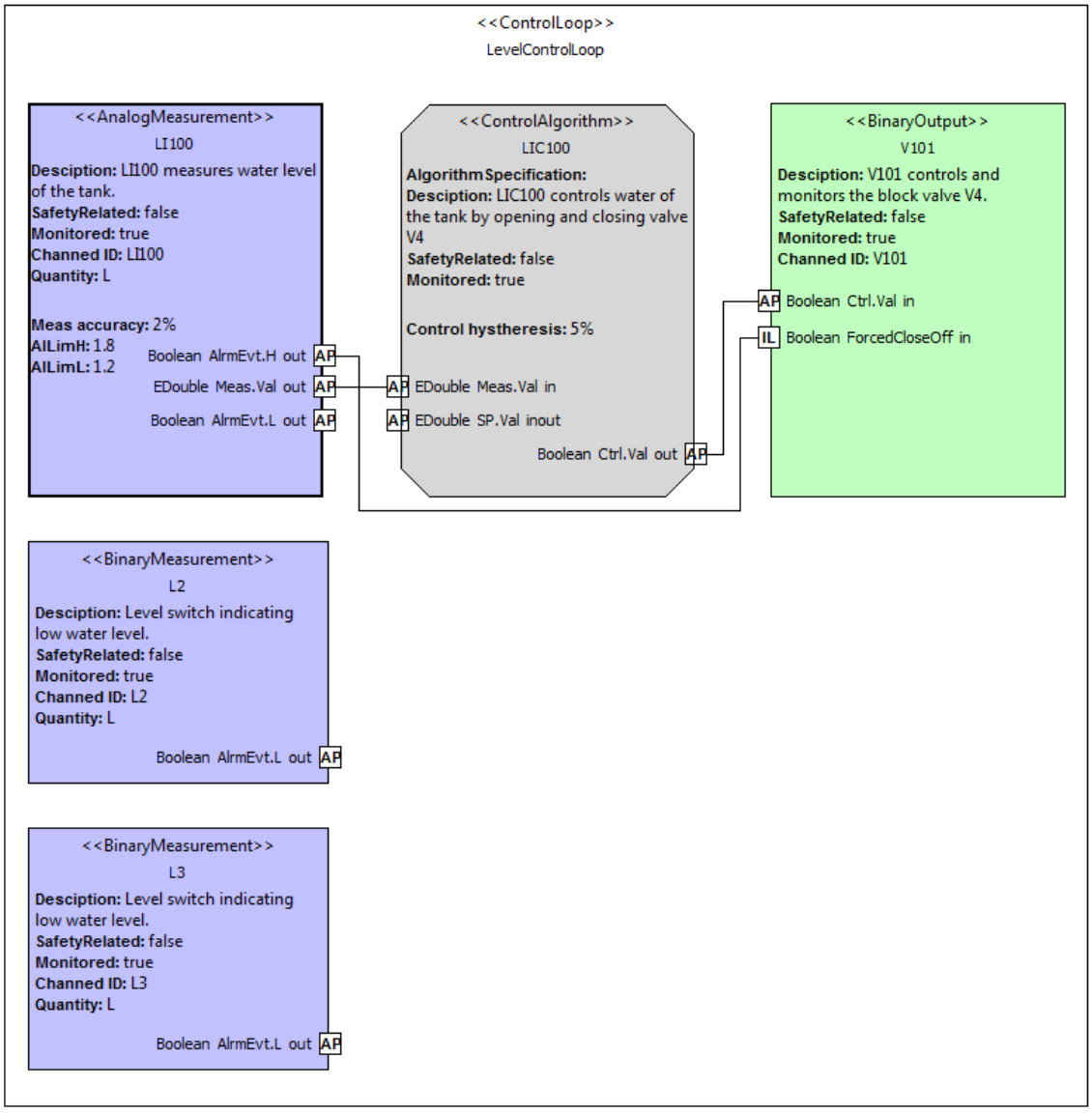
Kuva 5.11. *Esimerkkijärjestelmän mallimuunnoksessa generoitunut alustariippumaton mallin runko*

Kuvassa 5.11 voidaan havaita, että mallimuunnos vaatimusmallista alustariippumattomaan malliin on ollut varsin suoraviivainen. Tämä johtuu esimerkkiprosessin yksinkertaisuudesta. Vaatimuksista pinnan mittaukseen (LI100, L2 ja L3) on muodostunut Analog- ja BinaryMeasurement-elementtejä, eli toimintoja, joilla voidaan ottaa vastaan mittaussignaaleja. Vaatimuksesta säätää tankin pinnankorkeutta (LIC100) on muodostunut ControlAlgorithm, joka edustaa säätimen toiminnallisuutta. Vaatimuksesta ohjata veden virtausta tankkiin venttiilillä V4 (V101) on puolestaan muodostunut BinaryOutput-elementti, joka välittää binäärisen ohjaussignaalin toimilaitteelle. Generoitunutta alustariippumatonta mallia tarkennetaan ja muokataan seuraavassa kehitysprosessin askeleessa.

5.3.5. Automaatiotoimintojen tarkentaminen

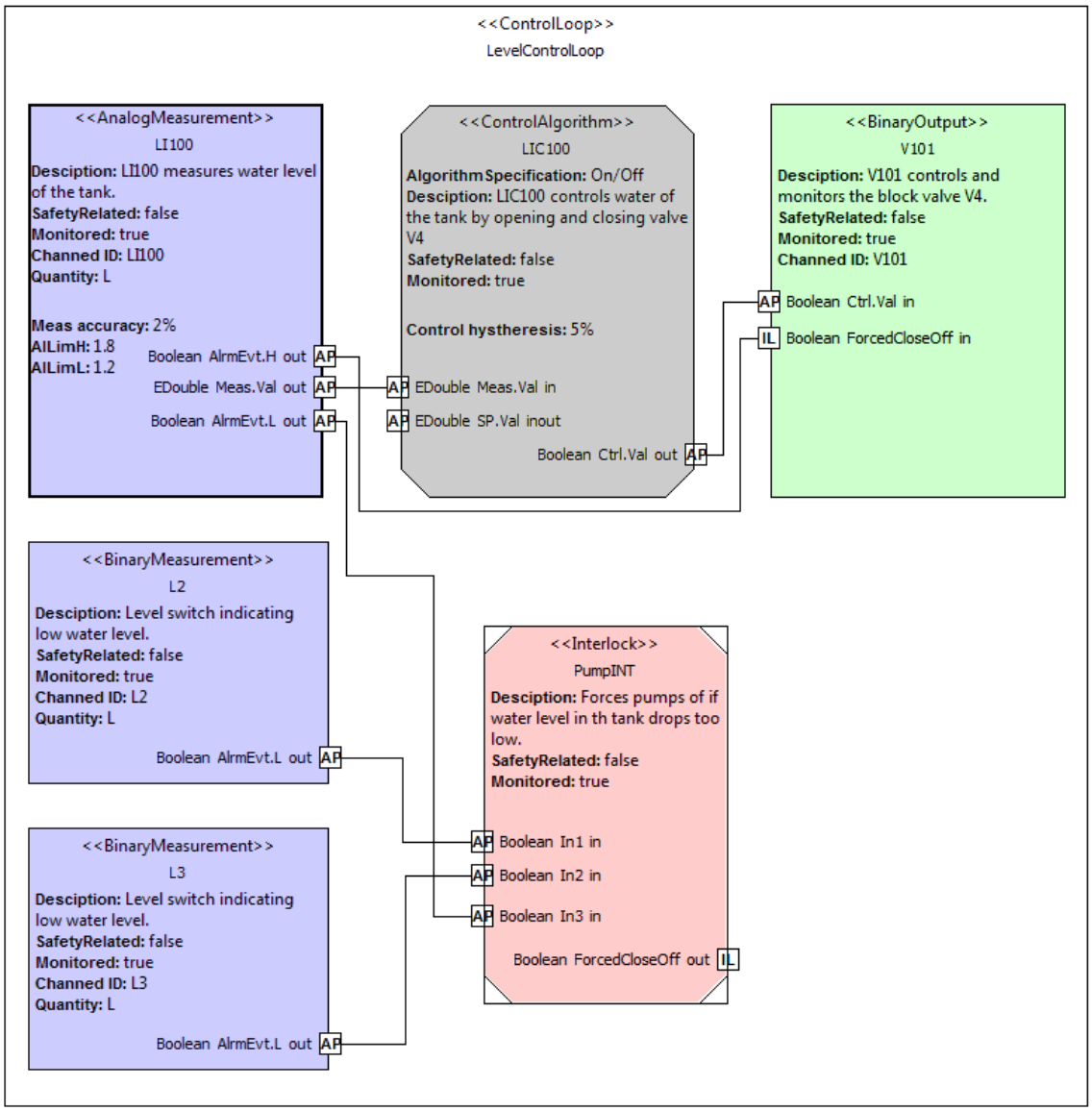
MDA-prosessin mukaisesti vaatimusmallin koostamisen jälkeen siirrytään alustariippumattoman mallin kehittämiseen. AUKOTON-kehitysprosessissa tämä tarkoittaa automaatiosovelluksen toiminnallisuuden ja rakenteen määrittelyä automaatiotoimintojen avulla. Käytännön kehitysprosessissa tämä konkretisoituu sovelluksen toiminnallisuuden ja rakenteen määrittelyllä automaatioprofiilin käsitteitä hyödyntäen. Alustariippumaton suunnitteluvaihe erottaa AUKOTON-kehitysprosessin automaatio-ohjelmistojen perinteisestä suunnitteluprosessista, jossa tätä suunnitteluvaihetta ei hyödynnetä ainakaan vastaavalla tavalla kuin tässä.

Suunnitteluaskeleen pääaktiviteetit suunnittelijan kannalta ovat alustariippumattoman mallin graafisen esityksen toteuttaminen ja järjestäminen, tarkentaminen, muokkaaminen, puuttuvien toiminnallisuuksien lisääminen ja alustariippumattoman rakenteen määrittely. Aktiviteetit ovat siis hyvin vastaavia kuin vaatimusmallinkin tuottamisessa. Kuvassa 5.12 on esitetty graafinen esitys kuvassa 5.11 esitetystä alustariippumattomasta esimerkkiprosessin mallista. Mallia ei ole vielä tässä tarkennettu tai muokattu lukuun ottamatta graafisen esityksen järjestämistä. Huomaa, että esimerkiksi toimintojen väliset yhteydet olivat mallissa valmiina transformaation jälkeen. Periaatteessa alustariippumaton malli on jo tällaisena riittävän tarkka seuraavaan suunnitteluaskeleeseen siirtymistä ajatellen, mutta tarkentamalla mallia tässä vaiheessa voidaan säästää työtä seuraavissa suunnitteluaskeleissa.



Kuva 5.12. *Esimerkkiprosessin alustariippumaton mallin järjestetty graafinen esitys*

Järjestetyn graafisen esityksen perusteella alustariippumatonta mallia tarkennetaan ja muokataan niin, että se mahdollisimman tarkasti esittäisi järjestelmän toiminnallisuuden ja rakenteen. Mallin tarkentaminen ja muokkaaminen on suunnitteluaskeleen aktiviteeteista tärkein ja eniten suunnittelijan luovaa työtä vaativa osuus. Suunnittelija täydentää mallia niin, että se automaatioprofiilin käsittein, kuvaa ohjelmiston toiminnan ja rakenteen alustariippumattomasti. Kuvassa 5.12 esitetyssä alustariippumattomassa mallissa voidaan havaita ainakin seuraavat tarkennus- ja lisäyskohteet: LIC100:n spesifikaatio puuttuu ja pumppujen P1-P3 lukituslogiikkaa ei ole mallinnettu. Tässä tapauksessa lukituslogiikka koostuu lukitustoiminnosta, joka on yhdistetty mittaussisääntulojen alarajahälytyksiin. Suunnittelija täydentää ja tarkentaa puutteelliset tiedot, jonka jälkeen esimerkkitapauksen alustariippumaton malli on valmis. Tarkennettu alustariippumaton malli on esitetty kuvassa 5.13.



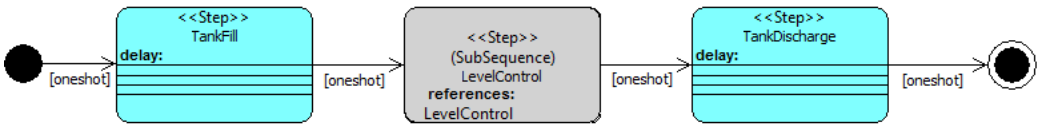
Kuva 5.13. *Esimerkkiprosessin tarkennettu alustariippumaton malli*

Kuvassa 5.13 esitetystä alustariippumattoman mallin graafisesta esityksestä voidaan havaita, että siihen on lisätty tarvittava lukituslogiikka mallinnettuna <<Interlock>> elementillä sekä LIC100:n algoritmin kuvaus, jotka puuttuivat alkuperäisestä mallista. Oleellisin näistä täydennyksistä on lukitustoiminto, koska se määrittää toiminnallisuutta. Pumppujen ohjausta ei ole kuvattu, koska se on esimerkkitapauksen ulkopuolella. Tankin pinnankorkeuden säätöpiiriin on kuitenkin lisätty logiikka, jolla pumppujen ohjauspiirille luodaan tieto mahdollisesta pinnankorkeuden liian matalasta arvosta.

Riippuen kehitettävästä sovelluksesta ja sen vaatimuksista, alustariippumattomassa suunnitteluaskeleessa voidaan tehdä paljon muitakin tarkennuksia ja lisäyksiä, kuten hälytysten, sekvenssien, tilojen, tapahtumien ja simulaatioiden mallinnusta. Erityisesti sekvenssien ja tapahtumien, hälytykset mukaan lukien, mallintaminen on tärkeää, koska ne kuvaavat järjestelmän toimintaa. Vaikka edellä tehty suunnittelutyö myös osaltaan kuvaa järjestelmän toimintaa, vasta sekvenssit kuvaavat sen tarkasti, siinä missä edellä tehty osa kuvaa paremminkin alustariippumatonta rakennetta.

Järjestelmän sekventiaalisesta toiminnasta ei aina ole olemassa lähtötietoja, mutta yleensä haluttu sekventiaalinen toiminta tulee esiin toimintakuvauksissa ja ajotapakeskusteluissa. Jos sekventiaalista toimintaa ei kuitenkaan ole määritelty, se jää suunnittelijan vastuulle. Esimerkkiprosessi on niin yksinkertainen, että se ei edes tarvitse sekvenssiä toimiakseen, mutta tässä sellainen määritellään, koska todellisessa suunnitteluprosessissa näin jouduttaisiin todennäköisesti tekemään. Esimerkkiprosessille tehty sekvenssi on viitteellinen, eikä sitä hyödynnetä kehitysprosessissa.

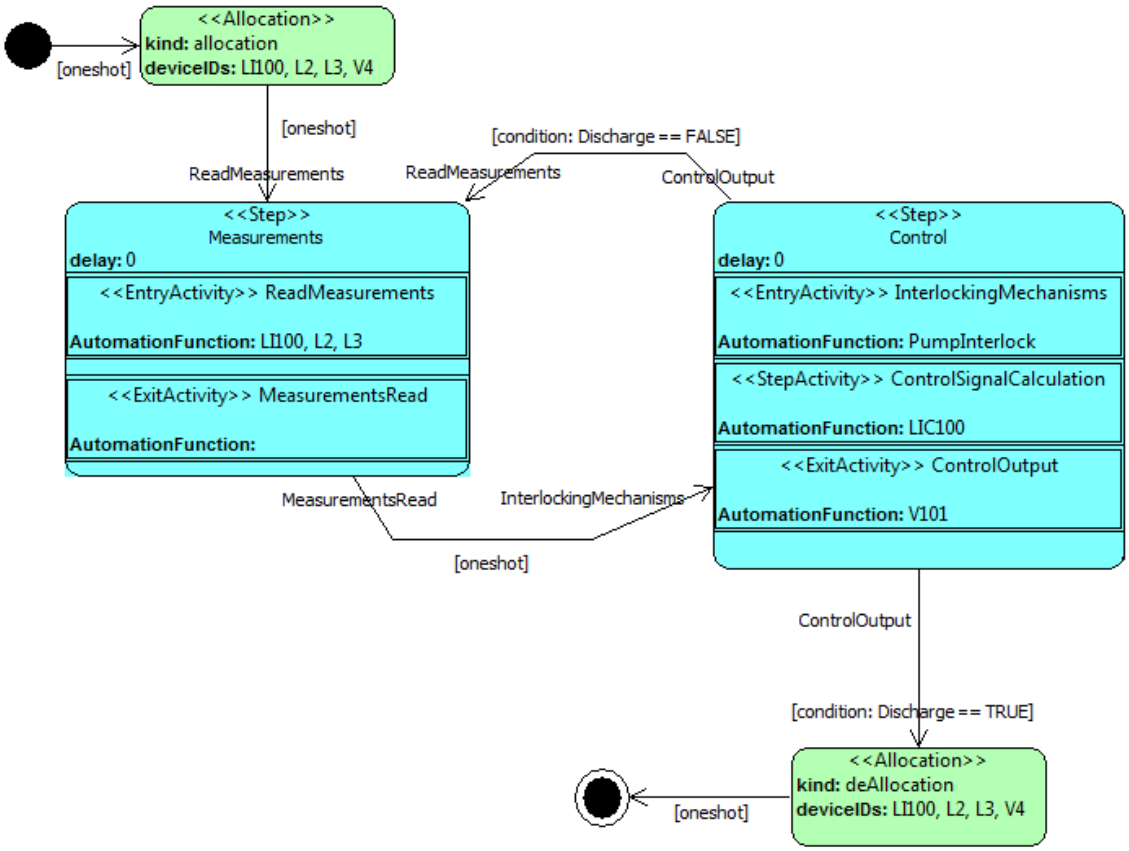
Esimerkkisekvenssi on kaksitasoinen kuvaus tankin toiminnasta. Kuvassa 5.14 on esitetty näistä korkeamman tason sekvenssi. Sekvenssi määrittelee tankin pinnansäädön vaiheet. Ensimmäisessä vaiheessa tankki täytetään (TankFill). Toisessa vaiheessa suoritetaan varsinainen pinnankorkeuden säätö (LevelControl), joka on tarkemmin kuvattu matalamman tason sekvenssillä. Tämä näkyy kaaviossa harmaana laatikkona, joka toimii myös linkkinä tarkemman sekvenssin määrittelevään kaavioon. Pinnankorkeuden säätö on vaiheista tärkein ja monimutkaisin, minkä vuoksi se on päätetty kuvata toisessa kaaviossa. Viimeinen vaihe on tankin tyhjentäminen (TankDischarge), minkä jälkeen sekvenssi päättyy ja se pitää käynnistää uudelleen esimerkiksi prosessin ylösajon yhteydessä.



Kuva 5.14. *Korkean tason sekvenssi esimerkkiprosessille*

Tarkastellaan seuraavaksi matalamman tason sekvenssiä (kuva 5.15), joka siis määrittää pinnankorkeuden säätövaiheen toiminnan (harmaa laatikko kuvassa 5.14). Matalan tason sekvenssiin siirrytään, kun TankFill askel on valmis. Sekvenssi alkaa laittei-

den allokoinnilla, eli laitteet varataan niitä ohjaavan sekvenssin käyttöön. Tämän jälkeen seuraa ensimmäinen varsinainen askel (Measurements), jossa luetaan anturien LI100, L2 ja L3 mittausarvot. Askeleesta poistutaan poistumisaktiviteetin (MeasurementsRead) kautta, jossa ei kuitenkaan tässä tapauksessa tehdä mitään. Poistumisaktiviteetti on tässä ainoastaan siirtymien, eli askelten välisten nuolien, oikeaoppista merkitsemistä varten. Seuraava vaihe (Control) alkaa lukitusrutiinin ajavalla esiaktiviteetilla (InterlockingMechanisms). Lukitusrutiini (PumpInterlock) tarkistavaa mittausarvot ja suorittaa tai purkaa lukituksen mittausten mukaisesti. Askeleen pääaktiviteetti (ControlSignalCalculation) on ohjaussignaalin muodostaminen. Tässä aktiviteetissa LIC100 muodostaa ohjaussignaalin sisääntulojensa perusteella. Viimeisessä vaiheen aktiviteetissa ohjaussignaali lähetetään venttiilille automaatiotoiminnon V101 toimesta. Kun vaiheen viimeinen aktiviteetti on saatu suoritettua, saavutaan haarautumispisteeseen, jossa päätetään jatketaanko vai lopetetaanko pinnankorkeuden säätö. Valinta riippuu muuttujan Discharge-arvosta, jota operaattori tai korkeamman tason ohjauslogiikka säätelee. Jos tankkia ei tyhjennetä, jatkuu sekvenssin suoritus mittausarvojen lukemisella. Jos taas tankki halutaan tyhjentää, allokoidut laitteet vapautetaan ja matalan tason sekvenssin suoritus päättyy, mutta korkean tason sekvenssi jatkuu TankDischarge-askeleella. Edellä selostettu matalan tason säätösekvenssi on esitetty graafisesti kuvassa 5.15.



Kuva 5.15. *Esimerkkiprosessin pinnankorkeuden säädön sekvenssi (matalan tason sekvenssi)*

Kuvan 5.15 mallia on yksinkertaistettu, eikä siinä hyödynnetä kaikkia profiilin tarjoamia mallinnuselementtejä, kuten lukituksia ja poikkeuksia. Automaatioprofiilin käsitteistö sallii monimutkaistenkin toiminnallisuuksien kuvaamisen tarkasti, mutta tässä mahdollisimman tarkan, mutta näin ollen erittäin monimutkaisen kuvaajan esittämistä ei katsottu tarpeelliseksi.

Alustariippumattomassa suunnitteluaskeleessa suunnittelija mallintaa järjestelmän alustariippumattomasti käyttäen edellä kuvatun kaltaisia malleja. Järjestelmän alustariippumaton rakenne ja osa toiminnasta mallinnetaan kuvissa 5.12 ja 5.13 esitellyllä tavalla. Alustariippumaton rakenne määrää, kuinka erilaiset automaatiotoiminnot koostuvat toisista automaatiotoiminnoista. Esimerkkiprosessissa säätöpiiriä esittävä automaatiotoiminto koostuu monista pienemmistä automaatiotoiminnoista (Kuva 5.13), kuten <<AnalogMeasurement>> ja <<ControlAlgorithm>> automaatiotoiminnoista. Tarkempi toiminnallisuus puolestaan mallinnetaan esimerkiksi kuvien 5.14 ja 5.15 mukaisesti, mikäli tälle on tarvetta. Alustariippumattoman mallinnusvaiheen toiminta on täysin verrannollinen MDA-prosessin alustariippumattoman mallin koostamiseen.

5.3.6. Automaatiotoiminnoista automaatiokomponentteihin

Kun alustariippumaton malli on osin tai kokonaan saatu valmiiksi, suunnittelu etenee alustariippuvaan vaiheeseen. MDA-prosessin mukaisesti siirtymä alustariippumattomasta alustariippuvaan malliin hoidetaan transformaatiolla. Tätä MDA-prosessin vaihetta pidetään sen monimutkaisimpana osana [18]. AUKOTON-kehitysprosessissa transformaatio alustariippumattomasta mallista alustariippuvaan on suunnittelijan kannalta mielenkiintoisin ja työläin, koska transformaatio on käyttäjäavusteinen. Tämä toisaalta yksinkertaistaa transformaation toteutusta, mutta samalla se tuo kehitysprosessiin manuaalisen työvaiheen. Transformaation käyttäjäavusteisuus on käytännön ratkaisu monimutkaisen transformaation suorittamiseksi. Vaikka tavoitteena on vähentää käyttäjän osuutta transformaation suorittamisessa, ei transformaatiota kokonaisuudessaan voida, eikä sitä ole mielekästä automatisoida kokonaan.

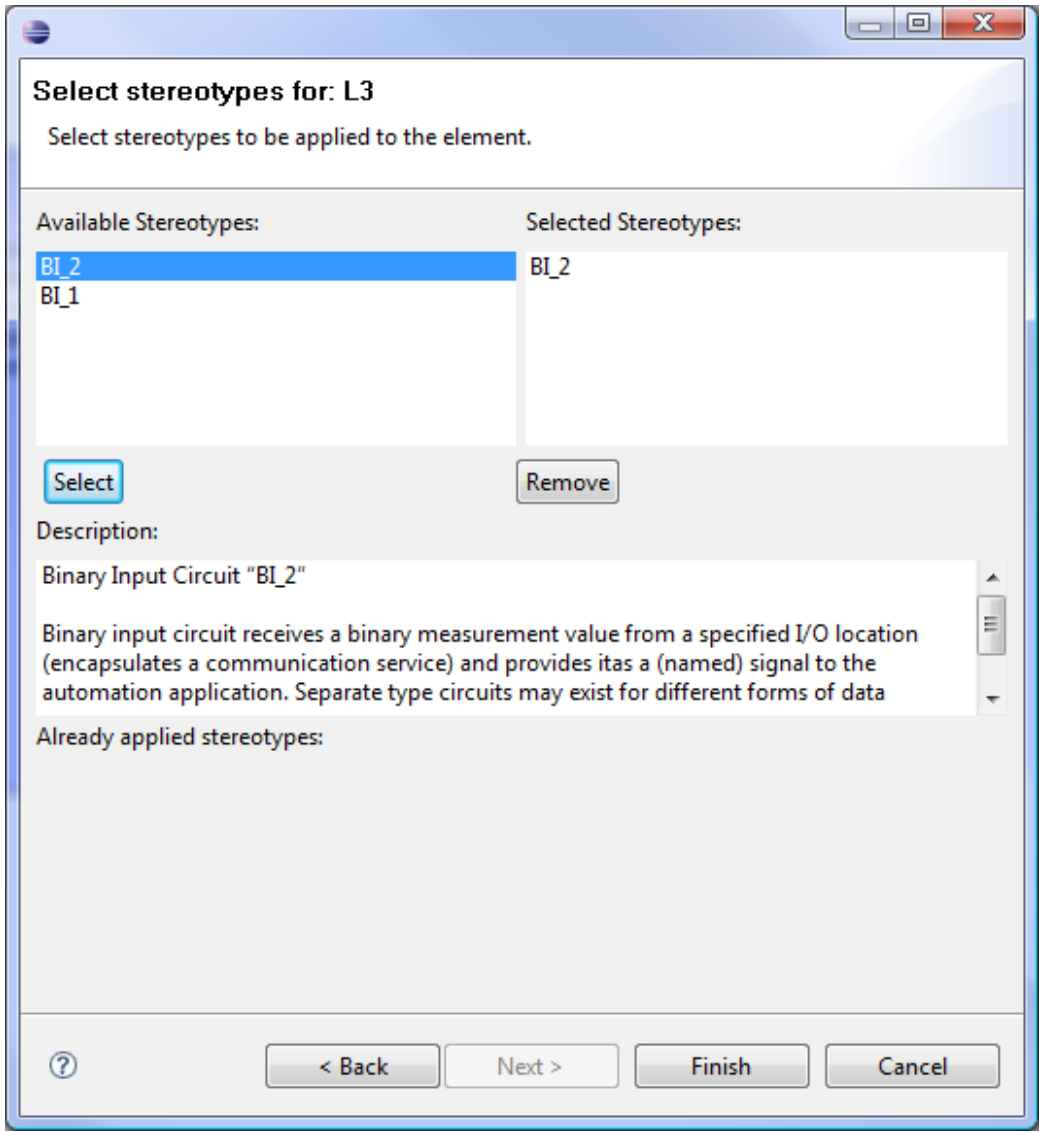
Transformaation pohjana on alustariippumaton malli tai sen osa, joka halutaan transformoida alustariippuvaan muotoon. Transformaatio suoritetaan lisäämällä transformoitavan alustariippumattoman mallin osan automaatiotoimintoihin alustakohtaisia stereotyyppejä, jotka on määritelty kohdealustan, esimerkiksi DCS-järjestelmän, mallissa. Vepsäläisen et al. mukaan stereotyypeillä automaatiotoiminto sidotaan toteutettavaksi jollakin kohdealustan toimilohkotyypillä. Stereotyypillä siis lisätään automaatiotoimintoon sellaisia ominaisuuksia ja parametreja, jotka sitä kohdealustalla vastaavalle toimilohkotyypille on määriteltävä. Osa ominaisuuksista ja parametreista on suunnittelun aikana spesifioitavia ja osa taas ajonaikana reaaliaikaisesti muuttuvia tietoja. [40]. Tarkastellaan seuraavaksi tarkemmin transformaation läpiviemistä käytännössä.

Transformaatio aloitetaan käynnistämällä AP-työkalusta wizard, jonka ohjauksessa automaatiotoiminnoille valitaan sopivat vastineet kohdealustan mallista. Ensin valitaan ne elementtityypit, joille kohdealustan stereotyyppejä halutaan lisätä. Yleensä ei ole tarpeen valita listalta kaikkia mahdollisia elementtityyppejä, koska esimerkiksi porteille

ei yleensä ole tarvetta lisätä alustariippuvia stereotyyppejä. Seuraavaksi valitaan se kohdealusta, jolla automaatiosovellusta tullaan ajamaan. Käytännössä valitaan kohdealustan profiili, joka määrittelee kohdealustan toimilohkotyyppejä vastaavat stereotyypit alustariippumattomille automaatiotoiminnoille.

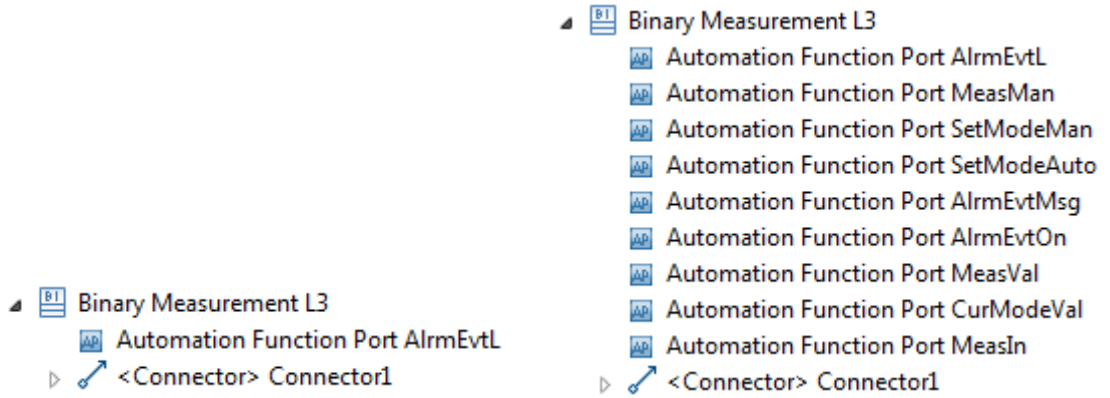
On syytä korostaa, että tähän asti suunnittelu on ollut lopullisesta ajoalustasta riippumatonta ja kehitetystä alustariippumattomasta mallista voidaan eri kohdealustan profiilien avulla erikoistaa eri alustoilla toimivia sovelluksia. Kun tarkennettavat automaatiotoiminnot ja käytettävä kohdealustan profiili on valittu, varsinainen stereotyyppien lisääminen alkaa.

Wizard käy läpi kaikki alustariippumattoman mallin valittua tyyppiä olevat automaatiotoiminnot ja käyttäjä valitsee avustajan tarjoamista vaihtoehdoista haluamansa stereotyypin kullekin automaatiotoiminnolle. Kuvassa 5.16 on esitetty transformaatiossa käytettävä wizard. Kuvassa nähdään yksinkertainen valintatilanne, jossa mittaussisääntulolle L3 (pintakytkin) ollaan valitsemassa sopivaa stereotyyppiä kohdealustan profiilista. Vaihtoehtoina ovat BI_2, BI_1 (kaksi erityyppistä binäärimittausta), joista suunnittelija on valinnut tyypin BI_2, koska se sopii kuvauksensa mukaan paremmin kyseiseen käyttötarkoitukseen. Muille automaatiotoiminnoille sopiva stereotyyppi valitaan vastaavasti.



Kuva 5.16. *Näkymä transformaatiossa käytettävään avustajaan (kuvakaappaus AP-työkalusta)*

Kun kaikille automaatiotoiminnoille on valittu alustakohtaiset stereotyypit, suunnittelija tarkistaa ja hyväksyy tehdyt valinnat. Hyväksynnän jälkeen wizard lisää stereotyypit järjestelmän malliin ja päivittää myös kaaviot niiden elementtien osalta, joille stereotyyppejä lisättiin. Lisäksi wizard lisää automaatiotoimintoihin stereotyyppien määrittelemät ominaisuudet ja tarvittavat portit, joita saattaa stereotyypin edustamasta toimilohkotyypistä riippuen olla suurikin määrä. Kuvassa 5.17 on esitetty järjestelmän malli mittaussisääntulon L3 osalta ennen ja jälkeen transformaation. Ennen transformaatiota automaatiotoiminnolla on vain yksi portti (Kuva 5.17a). Transformaatiossa lisätty alustariippuva stereotyyppi, joka sitoo toiminnon alustakohtaiseen toimilohkotyyppiin, on kuitenkin lisännyt automaatiotoimintoon huomattavan määrän portteja, joita toimilohkotyypin toteutus tarvitsee. Alustariippuvalla automaatiotoiminnon vastineella onkin yhdeksän porttia (Kuva 5.17b). Tämä on hyvä esimerkki siitä, että toiminnallisuutta voidaan yleensä mallintaa varsin vähäisellä määrällä portteja, mutta todellista toteutusta varten portteja tarvitaan usein paljon enemmän.



a) Ennen transformaatiota b) Transformaation jälkeen

Kuva 5.17. *Elementin L3 portit ennen ja jälkeen transformaation*

Transformaatiossa alustariippumattomat automaatiotoiminnot muuttuvat alustariippuviksi automaatiokomponenteiksi. Yleisin vastaavuus automaatiotoimintojen ja -komponenttien välillä on yksi yhteen vastaavuus. Tällöin kutakin alustariippumattoman mallin automaatiotoimintoa vastaa tarkalleen yksi automaatiokomponentti alustariippuvassa mallissa. Periaatteessa voi esiintyä tilanteita, joissa yksi automaatiokomponentti vastaa useampaa kuin yhtä automaatiotoimintoa tai päinvastoin. Nykyinen kehitysprosessi tosin sallii ainoastaan yksi yhteen tai yksi moneen vastaavuuden automaatiotoimintojen ja -komponenttien välillä, koska yhteen automaatiotoimintoon voidaan lisätä yksi tai useampia alustariippuvia stereotyyppejä. Esimerkkitapauksessa automaatiotoimintojen ja komponenttien vastaavuus on yksi yhteen, jolloin automaatiokomponentit muodostavat vastaavan hierarkian kuin automaatiotoiminnotkin.

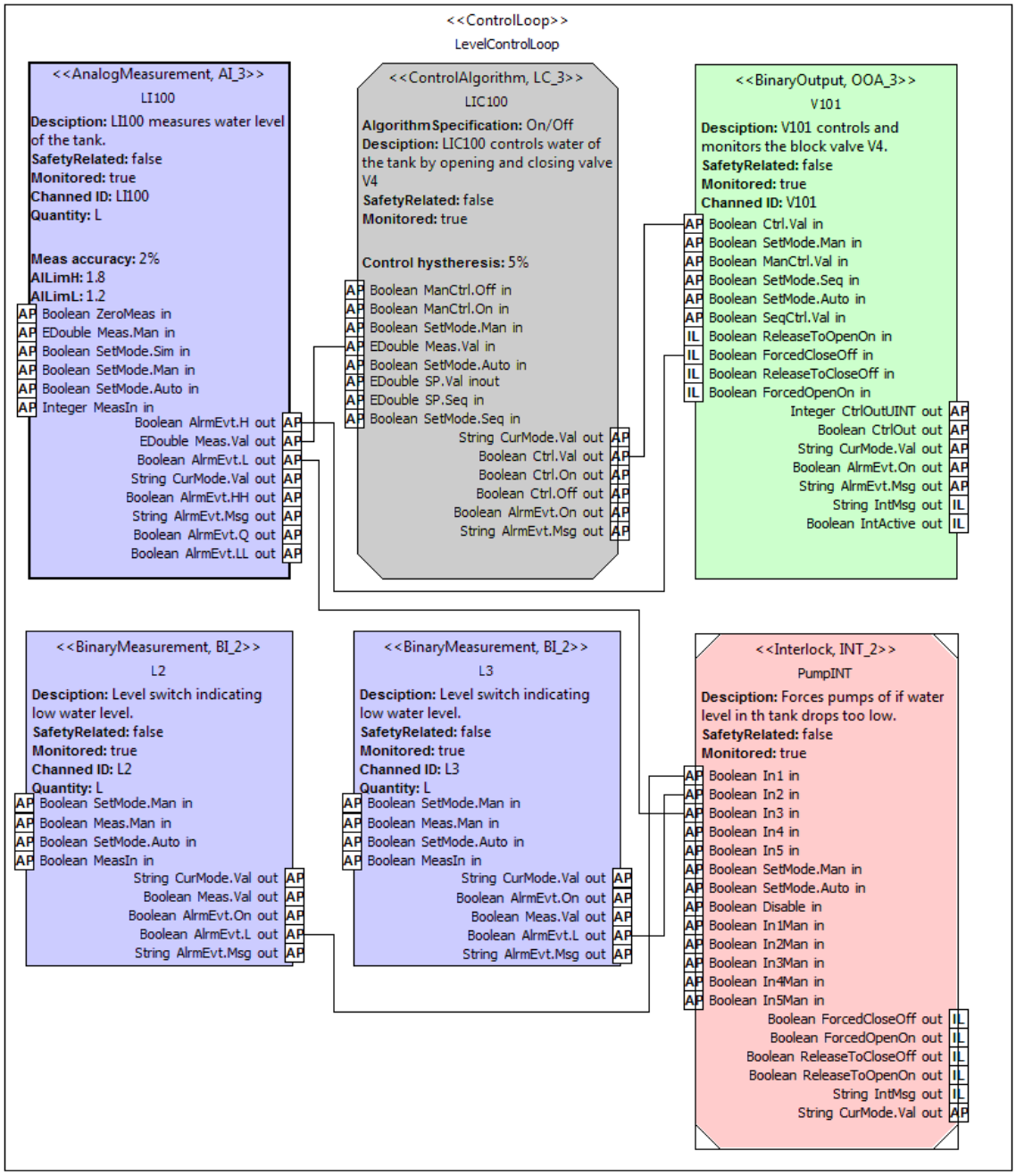
5.3.7. Automaatiokomponenttien tarkentaminen

Viimeinen AUKOTON-kehitysprosessin varsinaisista suunnitteluvaiheista on alustariippuva suunnittelu. Edellisessä vaiheessa alustariippumaton malli muunnettiin alustariippuvaksi käyttäjäavusteisessa transformaatiossa. Tässä suunnitteluvaiheessa alustariippuvaa mallia tarkennetaan kohti tilaa, jossa se mahdollisimman täydellisesti, alustariippuvat seikat huomioiden sekä sitoutuen alustan tarjoamiin ominaisuuksiin, palveluihin ja toimilohkotyyppeihin, kuvaa kehitettävän järjestelmän. Alustariippuvan vaiheen tärkeimmät suunnittelijan aktiviteetit ovat mallin graafisen esityksen järjestely transformaation jäljiltä, transformaatiossa muodostuneiden porttien yhdistäminen sekä sovelluskomponenttien rakenteen, allokoinnin, hajautusmallin ja kommunikoinnin määrittely sekä fyysisen laitteiston mallinnus, jolloin ohjelmisto voidaan sitoa fyysiseen laitteistoon [9]. Tätä kirjoitettaessa kehitysprosessissa ei suoriteta kaikkia mainittuja aktiviteetteja, vaan ainoastaan kaksi ensin mainittua. Muutkin aktiviteetit käydään kuitenkin tässä läpi.

Alustariippuvassa suunnitteluvaiheessa mallin graafisen esityksen järjestäminen poikkeaa hieman edellisissä vaiheissa nähdystä tavasta. Kuten edellä mainittiin, alustariippuva malli muodostuu alustariippumattoman mallin ”päälle”, jolloin järjestelmän

malliin ei luoda uutta pakettia alustariippuvalle osalle. Alustariippuva suunnittelu siis jatkuu suoraan samaa kaaviota hyödyntäen. Suunnittelijan tulee kuitenkin järjestellä transformaatiossa mallinnuselementteihin lisätyt portit niin, että ne ovat luettavissa.

Ensimmäinen suunnittelijan luovaa näkemystä vaativa aktiviteetti suunnitteluvaiheessa on transformaatiossa komponentteihin muodostuneiden toimilohkotyyppien tietovirtoja esittävien porttien kytkeminen toisiinsa niin, että näistä muodostuu mielekkäitä toiminnallisia kokonaisuuksia. Tämä vaihe muistuttaa nykyistä lähestymistapaa automaation ohjelmistosuunnitteluun, sillä tässä langoitetaan vastaavia toimilohkotyyppejä kuin nykyisissä alustariippuvissa suunnittelutyökaluissa kuten esimerkiksi Multiprogissa. Samalla alustariippumattomassa vaiheessa käytetyt ylimääräiset portit voidaan poistaa, mikäli ne eivät suoraan vastaa mitään alustariippuvan stereotyypin mukana tuomaa porttia. Näin mallin alustariippuviksi muunnetut toiminnot saadaan vastaamaan kohdejärjestelmän toimilohkotyyppejä. Kuvassa 5.18 on esitetty esimerkkitapauksen alustariippuva malli, joka on järjestelty ja tarvittavat porttien kytkennät on suoritettu.

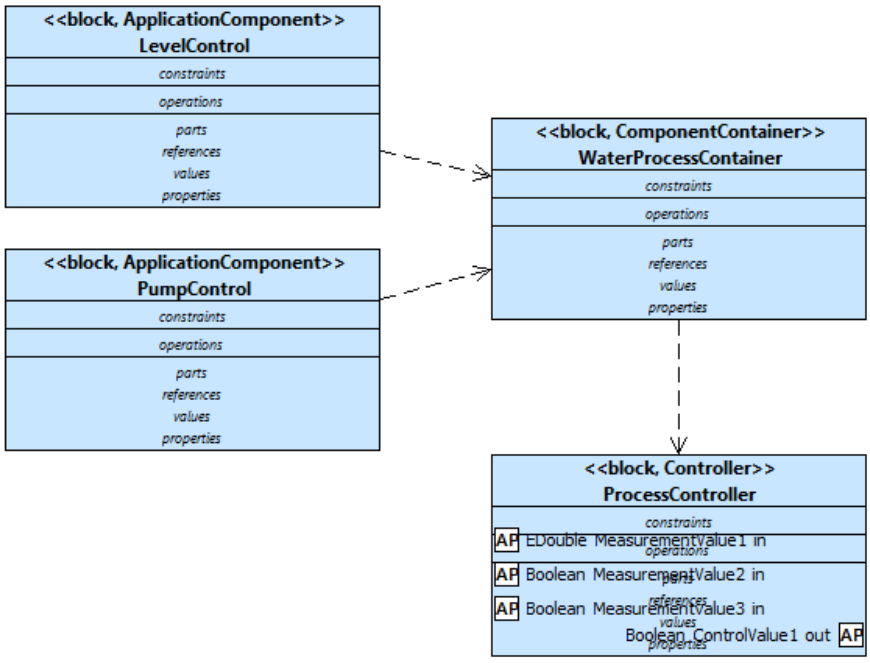


Kuva 5.18. *Esimerkkiprosessin järjestelty ja porttien sekä niiden yhteyksien osalta täydennetty alustariippuva malli.*

Seuraava looginen suunnitteluaktiviteetti on komponenttirakenteen määrittely. Tässä aktiviteetissä määritellään korkean tason automaatiokomponenttien väliset yhteydet. Määrittely tehdään sellaisten automaatiokomponenttien tasolla, jotka edustavat mielekästä suoritettavaa yksikköä kuten esimerkkitapauksen LevelControl, joka edustaa tankin pinnankorkeutta säätävää ohjelmakokonaisuutta (sovelluskomponentti). LevelControl-komponentti koostuu pienemmistä komponenteista, mutta koska ne yksin eivät muodosta mielekästä sovelluskomponenttia kokonaisjärjestelmän kannalta, ei niitä kannata ottaa mukaan rakenteen määrittelyyn, vaan ne sisällytetään LevelControl-komponentin sisäiseen rakenteeseen, joka puolestaan saadaan tässä tapauksessa suoraan

automaatiotoimintojen rakenteesta. Esimerkkitapauksen LevelControl-komponentti voidaan kokonaisjärjestelmän rakenteen tasolla yhdistää pumppuja ohjaavaan sovelluskomponenttiin, koska LevelControl-komponetti tarjoaa pumppuja ohjaavalle sovelluskomponentille sen tarvitsemaa tietoa. Seuraavat suunnitteluaktiviteetit perustuvat edellä kuvatulle mallin jakamiselle suoritettaviksi sovelluskomponenteiksi.

Kun järjestelmä on jaettu suoritettaviin osiin, eli sovelluskomponentteihin, voidaan näiden välinen hajautus ja kommunikointi määritellä. Vähänkään suuremmissa automaatiojärjestelmissä koko ohjausohjelmistoa ei pystytä ajamaan yhdellä prosessiasemalla, vaan ohjelmisto hajautetaan useammalle prosessointiyksikölle. Suunnittelija jakaa sovelluskomponentit säilöihin, jotka muodostavat yhdellä prosessorilla suoritettavan kokonaisuuden. Säilö voi tarjota sovelluskomponenteille ajoalustan palveluineen kuten esimerkiksi JRE- (Java Runtime Environment) tai .NET-alustat. Kukin säilö osoitetaan suoritettavaksi, eli allokoidaan yhdelle suoritinyksikölle. Kuvassa 5.19 on esitetty esimerkkijärjestelmän sovelluskomponenttien allokointi komponenttisäilöön ja säilön allokointi suoritinyksikölle, tässä tapauksessa säätimelle. Säädin on esitetty myös osana fyysisen laitteiston mallia, joka esitellään myöhemmin tässä luvussa. Kuvan esimerkkiin on otettu mukaan myös esimerkkiprosessin pumppuja P1-P3 ajava PumpControl-sovelluskomponentti, joka kuvan määritelmän mukaan suoritetaan samassa samalla suoritinyksiköllä kuin pinnankorkeutta säätävä LevelControl-komponenttikin.

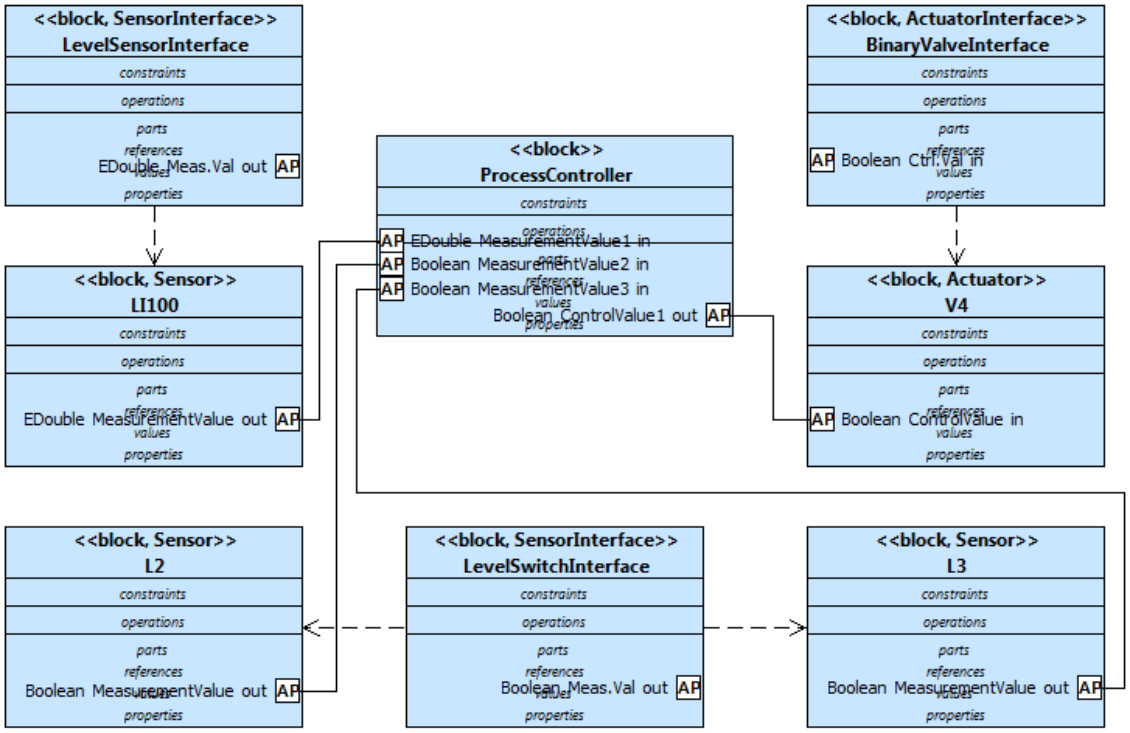


Kuva 5.19. *Esimerkkijärjestelmän sovelluskomponenttien allokointi komponenttisäilöön ja suoritusyksikölle*

Koska järjestelmä voidaan hajauttaa eri suoritinyksiköille, suunnittelijan tulee ottaa kantaa myös sovelluskomponenttien kommunikointiin sekä rinnakkaisuudesta ja toimintojen samanaikaisuudesta aiheutuviin ongelmiin. Vaikka toistensa palveluita eniten tar-

vitsevat sovelluskomponentit voidaankin yleensä laittaa suoritukseen samaan komponenttisäilöön, lähes aina on tarve myös eri säilöissä ajettavien sovelluskomponenttien väliselle kommunikaatiolle. Pyrkimyksenä on kuitenkin minimoida suoritinyksiköiden välinen liikenne, koska se on hitaampaa ja epäluotettavampaa kuin suoritinyksikön sisäinen liikenne. Automaatioprofiili tukee useita eri hajautusmalleja kuten client-server, proxy ja broker -mallit, joista suunnittelija voi valita käyttötarkoitukseen parhaiten sopivan. Esimerkkitapauksessa hajautusta ei tarvita, koska esimerkin sovelluskomponentit voidaan suorittaa yhdellä suoritinyksiköllä. Esimerkkiprosessi on kuitenkin osa suurempaa järjestelmää jonka laajuudessa hajautusta jo tarvittaisiin.

Kuten edellä mainittiin, myös järjestelmän fyysinen laitteisto mallinnetaan. Laitteiston mallin avulla ohjelmisto sidotaan käytettävään laitteistoon laiterajapintoja hyödyntäen. Muistetaan, että alustariippumattomassa vaiheessa laitteita kuvattiin ainoastaan IO-toimintojen kautta. Laitteiston mallissa esitetään automaatio- ja kommunikointilaitteet sekä niiden rajapinnat ja fyysiset kytkennät. Fyysisiä kytkentöjä on kahta tyyppiä, joista ensimmäinen on passiivinen kytkentä eli esimerkiksi jännitesignaali parikaapelissa ja toinen on aktiivinen kytkentä eli kenttäväylä. Osa fyysisestä laitteistosta voidaan esittää jo lähtötiedoissa. Tämä tieto voitaisiin tuoda suunnittelijan käytettäväksi laitteiston mallinnusvaiheessa toteuttamalla lähtötietoja tuova AP-työkalun liitännäinen, joka suoraan lähtötietojen perusteella rakentaisi laitteiston mallin tai osan siitä. Tällaista toiminnallisuutta ei kuitenkaan tätä kirjoitettaessa ole vielä implementoitu AP-työkaluun. Esimerkkitapauksen fyysisen laitteiston malli on esitetty kuvassa 5.20. Laitteistoon kuuluu analoginen pinta-anturi (LI100), kaksi pintakytkintä (L2 ja L3), on/off-venttiili (V4) ja tankin pinnankorkeutta ohjaava säädin (ProcessController). Säätimen ja muiden laitteiden välinen yhteys on toteutettu yksinkertaisella parikaapelilla tai vastaavalla, koska yhteyksiä ei ole merkitty kommunikaatioväyliksi.



Kuva 5.20. *Esimerkkiprosessin fyysisen laitteiston malli pinnansäädön osalta*

Alustariippuvan suunnitteluaskeleen päämäärä on tuottaa järjestelmästä niin tarkka malli, että sen perusteella pystytään generoimaan järjestelmän sovelluskoodi. Suunnitteluaskeleen aktiviteetit ovat iteratiivisia, eivätkä ne välttämättä etene esitetyssä järjestyksessä. Varsinkin komponenttien hajautus, kommunikointi ja fyysisen laitteiston mallinnus saattavat olla suuresti riippuvia toisistaan, jolloin suunnittelusta muodostuu iteratiivinen prosessi, jossa kutakin osa-aluetta vuorollaan tarkennetaan.

5.3.8. Automaatiokomponenteista alustakohtaiseksi suoritettavaksi sovellukseksi

Lopullinen alustariippuva malli kuvaa toteutettavan järjestelmän ottaen huomioon kohdealustan. Alustariippuvaa mallia ei kuitenkaan voida käyttää sellaisenaan, koska mikään ajoalusta ei toistaiseksi tue AP-työkalulla tuotettua mallia ajettavana lähdekoodinaan[6]. Ei ole myöskään realistista odottaa, että tällaista tukea ainakaan lähitulevaisuudessa olisi saatavilla. Tästä syystä järjestelmästä luotu malli tulee transformoida kohdealustalle sopivaan muotoon. Transformaatio suoritetaan AP-työkaluun liitetyllä laajennuksella, joka pystyy muuntamaan alustariippuvassa mallissa käytetyt mallinnuselementit kohdealustan ymmärtämään muotoon. Kohdealusta voi olla esimerkiksi jokin DCS-alusta (Distributed Control System).

Alustariippuva malli voidaan joko transformoida suoraan ajettavaksi koodiksi tai alustakohtaisen työkalun ymmärtämään muotoon ja tästä edelleen ajettavaksi koodiksi.

[6] Mallipohjaisuuden lopullinen päämäärähän oli pelkkien mallien käyttö niin, että lähdekoodia ei tarvita ollenkaan, vaan ajoympäristölle voidaan antaa pelkkä malli, jonka perusteella ohjelmaa ajetaan.

On selvää, että valmiin (konekielisen) sovelluskoodin tuottaminen suoraan AP-työkalun liitännäisellä on huomattavan monimutkaista, joskin mahdollista. Helpompi ratkaisu onkin tuottaa alustariippuvasta mallista alustakohtaisen työkalun ymmärtämä tiedosto, jossa alustariippuvat elementit ja niiden yhteydet on korvattu alustan käyttämillä toimilohkotyypeillä. Tämä on varsin suoraviivainen operaatio, koska alustariippuva malli on erikoistettu alustan toimilohkotyyppejä vastaavilla stereotyypeillä. Stereotyyppien ansiosta toimilohkotyyppien tarvitsemat tietovirrat ja parametrit on alustariippuvissa mallielementeissä määritelty porttien ja ominaisuus-arvo -parien muodossa. Tässä luvussa koodingeneroinnilla tarkoitetaan kumpaa tahansa edellä mainituista toimenpiteistä, vaikka sanan varsinaisessa merkityksessä koodingeneroinnista ei olisikaan kyse.

Käytännössä koodingenerointi on automaattinen transformaatio, johon käyttäjän ei tarvitse osallistua. Suunnittelija vain valitsee sen osan alustariippuvasta mallista (yleensä koko mallin), josta koodia halutaan generoida. Tämän jälkeen valitaan sen alustan transformaattori, jolle koodi halutaan generoida. Generointi suoritetaan ja generaattori antaa ilmoituksen generoinnin onnistumisesta tai aiheutuneista ongelmista.

Koodingeneroinnin haasteet liittyvät tuotetun koodin kattavuuteen. Rauhamäen et al. mukaan generoinnin tavoitteena ja onnistumisen edellytyksenä on tuottaa koodia, jota suunnittelijan ei tarvitse käsin muokata tai täydentää. AUKOTON-kehitysprosessin lähestymistapa ongelmaan on generoida koodista vain se osa joka varmasti pystytään tuottamaan virheettömästi annetun mallin pohjalta. Yleensä tämä osa kattaa niin sanotun bulk-tiedon, kuten prosessiliitäntöjen konfiguraation tai muuta alustariippuvasta mallista yksinkertaisesti pääteltävää tietoa. [9]. Yksinkertaisessa tapauksessa, kuten tässä käsitellyn esimerkin puitteissa, on kuitenkin mahdollista generoida kokonaisuudessaan toimiva toteutus.

5.3.9. Seuraavat työvaiheet

Koodingeneroinnista ei yleensä saada lopputuloksena valmista ajettavaa sovelluskoodia, vaan lähdekoodia esimerkiksi C tai Java-kielellä tai järjestelmän kuvaus toimilohkotasolla alustaspesifin työkalun ymmärtämässä muodossa. Käytännössä toimilohkotason kuvaus tulee todennäköisesti olemaan yleisempi vaihtoehto yksinkertaisemman toteutuksensa ansiosta. Varsinainen ajettava ohjelmisto tulee näin ollen tuottaa käännöstyökaluketjulla tai alustaspesifillä työkalulla, kuten Multiprog:illa tässä esimerkissä. Multiprog on IEC 61131-3 [41] ohjelmointikieliä hyödyntävä ohjelmointialusta, jolla voidaan muun muassa generoida lopullinen ajettava sovelluskoodi ja ladata se laitteille ajettavaksi [34]. Muita vaihtoehtoja ovat esimerkiksi DCS-ohjelmointiympäristöt kuten FbCAD, joka on karkeasti Multiprog:ia vastaava työkalu MetsoDNA alustalle [42].

Tässä vaiheessa suoritetaan generoidun koodin mahdollinen muokkaaminen ja täydentäminen. Vaikka koodin muokkaaminen ja tarkentaminen ovatkin ehkä helpoin tie sovelluksen toimintaan saattamiseksi, pitäisi mallipohjaisen kehitysprosessin periaatteiden mukaisesti ongelmaa etsiä ensisijaisesti muodostetusta järjestelmän mallista ja transformaattoreista.

5.4. Mallipohjaisen lähestymistavan etuja ja haittoja automaatiosuunnittelun kannalta

Esitetty kehitysprosessi avaa mahdollisuuksia hyödyntää kertaalleen tehtyä suunnittelutyötä uudelleen eri projekteissa ja erilaisilla alustoilla. Perinteisessä automaatiosuunnittelussa alustasuuntautuminen alkaa näkyä jo aikaisessa vaiheessa projektia. Tähän itse asiassa jopa pyritään, jotta saavutettaisiin yhtenäinen käsitteistö projektiosapuolten kesken, mutta käytännössä alustariippuviin ratkaisuihin ajaudutaan muun muassa tiettyjen laitteiden pitkän toimitusajan takia. Tällöin voidaan saavuttaa kustannussäästöjä, koska suunnittelussa voidaan hyödyntää suoraan kohdealustan piirteitä valitsemalla esimerkiksi positiotunnukset kohdealustan kannalta sopivasti. Aikainen alustasuuntautuminen suunnitteluprojektissa vie pohjaa uudelleenkäytettävyydeltä, koska suunnitteluratkaisuja ei pystytä siirtämään projektista ja alustalta toiseen muuten kuin suunnittelijan tietämyksenä ja korkean tason ratkaisumalleina. AUKOTON-kehitysprosessissa korkean tason ratkaisumallit luodaan vaatimusmallissa ja alustariippumattomassa mallissa, joten niiden uudelleenkäytölle on edellytyksiä millä tahansa alustalla ja eri projekteissa.

Edellä tuli ilmi, että eri suunnitteluosapuolten välillä ei juuri hyödynnetä yhtenäistä automaation toimialakohtaista käsitteistöä, vaan projekteissa kiirehditään alustariippuvaan suunnitteluun, jotta edes sen kautta päästään käyttämään samoja käsitteitä. Mallipohjainen kehitysprosessi vastaa osaltaan myös tähän ongelmaan, koska sen käyttö vaatii toimialakohtaisen käsitteistön hyödyntämistä. Kohdassa 4.3 esitelty UML-automaatioprofiili sisältää AUKOTON-kehitysprosessin hyödyntämän mallinnuskäsitteistön, jota voidaan hyödyntää laajemmaltikin automaation suunnitteluprosessissa ja myös eri suunnitteluosapuolten välillä.

Automaation ohjelmistosuunnittelu on perinteisesti ollut dokumenttisuuntautunutta, eli kehitysprosessin lähtökohtana ovat olleet edeltävissä vaiheissa tehdyt järjestelmää kuvaavat dokumentit. AUKOTON-kehitysprosessi pyrkii tehostamaan edellisissä suunnitteluvaiheissa tuotettujen sähköisten lähtötietojen hyödyntämistä luomalla mahdollisuuksia niiden tuomiseen ohjelmistosuunnittelutyökaluun ilman manuaalista tiedon siirtämistä. Näin voidaan parantaa ohjelmistosuunnittelun integroitumista osaksi kokonaissuunnitteluprosessia. Integroituvuutta nostaa osaltaan sekin, että lopullisesta järjestelmää kuvaavasta mallista voidaan generoida koodia käytettäväksi alustaspesifeillä työkaluilla. AUKOTON-kehitysprosessissa hyödynnetään tiedon automaattista muuntamista myös kehitysprosessin eri vaiheiden välillä. Pyrkimyksenä on näin minimoida manuaalinen tiedonsiirto myös ohjelmiston suunnitteluprosessin aikana.

Mallipohjaisuuden keskeinen idea on käyttää malleja järjestelmän kuvaamiseen aina vaatimuksista tarkkaan alustariippuvaan malliin asti. Mallit ovat pitkälti itsedokumentoivia (tämä vaatii erittäin vakiintuneen käsitteistön), mikä luo edellytyksiä vähentää projektin aikana tehtyjen asiakirjadokumenttien määrää. Samalla pystytään vähentämään eri dokumenttiversioiden ylläpitotyötä, koska viittauksia eri dokumenttiversioiden välillä ei ole yhtä paljon kuin perinteisessä kehitysprosessissa.

Viimeisenä hyötynäkökohtana mallipohjaisesta lähestymistavasta mainittakoon paremmat mahdollisuudet säilyttää tehtyjen suunnittelupäätösten jäljitettävyys. Koska koko automaatiosovellus mallinnetaan vaatimuksista alustariippuvaan malliin asti samalla työkalulla, suunnittelupäätösten jäljitettävyys on helpommin kuljetettavissa järjestelmän mukana. Tieto suunnitteluratkaisuiden alkuperästä ja toisaalta niiden perusteella tehdyistä ratkaisuista voidaan säilyttää osana mallia.

Ehkä selkein haitta uudenlaisesta suunnitteluprosessista on se, että mallipohjaiset menetelmät ja UML eivät ole olleet automaatioalalla kovin yleisessä käytössä. Näin ollen automaatiosuunnittelijoilla ei välttämättä ole riittäviä esitietoja kehitysprosessin hyödyntämiseen. Tämä yhdistettynä automaatioalan konservatiivisuuteen ei ainakaan helpota suunnitteluprosessin mahdollisuuksia nousta yleisesti hyväksytyksi ja käytetyksi lähestymistavaksi automaatiosuunnitteluun. Tästä syystä tarvitaan ohjeistusta, jolla uuden kehitysprosessin menetelmät saadaan ”maadoitettua” nykyisiin suunnittelukäytäntöihin.

6. KEHITYSPROSESSIN OHJEISTUS

Tämän diplomityön tarkoituksena oli toteuttaa ohjeistus AUKOTON-projektissa kehitettävälle mallipohjaiselle automaatiosovellusten kehitysprosessille. Ohjeistus kattaa automaatioprofiilin käsitteistön, kaaviotyypit, kehitysprosessin periaatteet ja automaatioprofiilin käytön automaatiojärjestelmän mallintamisessa. Ohjeistus ei ole tarkoitettu AP-työkalun käyttöohjeeksi, mutta AP-työkalua ja sen käyttöä sivutaan ohjeistuksessa. Myöskään UML periaatteineen ja käytäntöineen ei varsinaisesti kuulu ohjeistuksen ydinalueeseen, mutta sitäkin sivutaan ohjeistuksen tietyissä kohdissa.

Työt automaatiosovellusten mallipohjaisen kehitysprosessin ohjeistuksen ja AUKOTON-projektin parissa aloitettiin toukokuussa 2008. Ohjeistuksen toteutuksen suunnittelua aloitettaessa itse projekti oli ollut käynnissä muutamia kuukausia, mutta töitä etenkin AP-työkaluun liittyen oli tehty jo kauan ennen projektin virallista aloitusajankohtaa. Suurin haaste ohjeistuksen toteuttamisessa on ollut tekijän käytännön kokemuksen puute automaatiosuunnittelusta. Ohjeistusta tehtäessä on kuitenkin pyritty asettumaan UML:ää ja mallipohjaisia menetelmiä tuntemattoman automaatiosuunnittelijan näkökulmaan ja toteuttaa ohjeistus siten, että se mahdollisimman hyvin palvelisi juuri heidän tarpeitaan.

Tässä luvussa esitellään niitä käytännön tuotoksia, joita ohjeistukseen on tähän mennessä saatu toteutettua. Luku aloitetaan perustelemalla ohjeistuksen tarpeellisuus ja esittelemällä sen kohderyhmä ja tarkoitus. Tämän jälkeen siirrytään varsinaiseen tuotettua ohjeistusta esittelevään osaan, jossa käydään läpi automaatioprofiilin ja automaatio-ohjelmistojen mallipohjaisen kehitysprosessin kirjallinen ohjeistus ja AP-työkaluun integroidut käyttäjää avustavat toiminnot ja ohjeet.

6.1. Ohjeistuksen tarpeen perustelu ja kohderyhmä

Aikanaan automaatio ei ollut ohjelmistokeskeistä vaan se toteutettiin pääasiassa releillä tai analogisella elektroniikalla. Tällöin automaatioinsinöörien koulutuksessakaan ei juuri, jos ollenkaan, panostettu ohjelmistotekniikan opettamiseen. Tämä näkyy nykyisinkin automaatioinsinöörien ja suunnittelijoiden osin puutteellisena ohjelmistotekniikan osaamisena ainakin mallipohjaisen kehityksen ja UML:n kohdalla. Niistä päivistä, jolloin automaatio toteutettiin laitteistopohjaisesti, on kuitenkin aikaa ja nykyisin automaatiojärjestelmät ovat kasvavassa määrin ohjelmistokeskeisiä [8]. Nykyään ohjelmistotekniikan tärkeys alalla on huomioitu myös automaationinsinöörien koulutuksessa.

Koska UML:n ja mallipohjaisten menetelmien tuntemus automaatioalalla on vielä paikoin kohtalaisen heikkoa, on diplomityössä toteutettu ohjeistus kohdistettu ensisijaisesti nykyisille automaatiosuunnittelijoille, joilla ei ole kokemusta mainituista teknii-

koista ja menetelmistä. Ohjeistuksen pyrkimyksenä on ollut maadoittaa kehitysprosessin työnkulku ja käsitteistö nykyisiin suunnittelukäytäntöihin. Tavoitteena on ollut kuvata käsitteistöä ja kehitysprosessia käyttäjälähtöisesti siten, että UML:ää ja mallipohjaista kehitystä tuntematonkin henkilö pystyy sen avulla omaksumaan käsitteistön käytön mallintamisessa, ymmärtämään kaavioita ja suorittamaan yksinkertaisen järjestelmän mallinnuksesta ohjeistukseen tukeutuen. Ohjeistus ei luonnollisesti pysty korvaamaan kehitysprosessissa sovellettujen käsitteiden ja menetelmien koulutusta, mutta sen pitäisi tuoda kehitysprosessi käsitteineen riittävän lähelle nykyisiä suunnittelukäytäntöjä, jotta suunnittelija voisi ohjeistuksen avulla selviytyä järjestelmän mallipohjaisesta suunnittelusta.

Toinen ohjeistuksen kohderyhmistä ovat (automaatioalan) opiskelijat. AUKOTON-kehitysprosessin tulevaisuuden kannalta on tärkeää, että sitä voidaan hyödyntää automaatioinsinöörien koulutuksessa jo ennen työelämään siirtymistä. Tällöin heillä on ymmärrystä vastaavantyyppisen kehitysprosessin periaatteista ja kokemusta sen soveltamisesta. On helpompaa ohjata kehitysprosessin oikeaoppiseen käyttöön sellainen henkilö, jolla ei ole voimakasta ennakkokäsitystä suunnittelukäytännöistä. Kokemattomuus automaatiosuunnittelussa on tietenkin opetuksen kannalta oma haasteensa. Toisaalta nykyisillä automaatioalan opiskelijoilla on suhteellisen kattava osaaminen myös ohjelmistotekniikan alueelta, mikä saattaa helpottaa AUKOTON-lähestymistavan omaksumista.

6.2. Ohjeistuksesta yleisesti

Tässä diplomityössä toteutettu ohjeistus käsittelee automaatioprofiilia (käsitteistö) ja automaatio-ohjelmistojen mallipohjaista kehitysprosessia (AUKOTON-kehitysprosessi). Kuten edellä jo kävi ilmi, ohjeistus ei ole tarkoitettu AP-työkalun käyttöohjeeksi, vaan se pyrkii ohjeistamaan automaatioprofiilin ja kehitysprosessin oikeanlaiseen käyttämiseen ja tulkintaan. Ohjeistuksessa tulee esiin joitakin työkalun käyttöön liittyviä seikkoja niiltä osin kun ne sopivat tai kuuluvat ohjeistuksessa käsiteltävän asian luonteeseen. Vaikka ohjeistuksella ei pyritäkään ohjeistamaan työkalun käyttöä, on AP-työkalulla luonnollisesti suuri rooli ohjeistuksessa. AP-työkalua on käytetty ohjeistuksen esimerkkien toteuttamiseen ja sen yhteyteen on toteutettu työkalusidonnaista ohjeistusta.

Ohjeistus on toteutettu englanniksi. Englannin kieleen päädyttiin, jotta mahdollisimman suuri osa ohjeistuksen käyttäjäkunnasta voisi hyödyntää sitä. Mahdollinen suomenkielinen versio olisi kuitenkin pitänyt jossain vaiheessa kääntää englanniksi, jotta se olisi myös kansainvälisen tutkija- ja yritysyhteisön käytettävissä.

Ohjeistuksessa on hyödynnetty kahta erilaista ohjeistusmuotoa, jotka ovat kirjallinen ohjeistus sekä työkaluun integroidut ohjeet ja avusteet. Kirjallinen ohjeistus on tavallinen asiakirjadokumentti. Työkaluun integroidut ohjeet ja avusteet puolestaan ovat työkalussa olevia, käyttäjää avustavia ja ohjeistavia toiminnallisuuksia, sekä työkalun oh-

jeselaimella käytettävä versio kirjallisesta ohjeistuksesta. Ohjeistuksen eri muodot ovat periaatteineen ja sisältöineen kuvattu kohdissa 6.3 ja 6.4.

Miten sitten juuri tällaiseen ohjeistukseen päädyttiin? Ohjeistuksen toteutuksen alkuvaiheessa automaatioprofiilin käsitteiden ohjeistus oli luonnollinen aloituspiste, koska profiilin käsitteiden ymmärtäminen on oleellista mallintamisen kannalta, joka taas on koko kehitysprosessin ydin. Tästä syystä automaatioprofiilin käsitteistön ohjeistus on suuressa roolissa ohjeistuksen kokonaisuuden kannalta. Samalla tutustuttiin profiilin käsitteistöön, mikä osaltaan auttoi koko kehitysprosessin ymmärtämisessä. Automaatioprofiilin käsitteiden ohjeistukseen on kulunut suurin osa ohjeistuksen tekemiseen käytetystä ajasta.

Ajatus ohjeistuksen integroimisesta AP-työkaluun on peräisin TTY:llä työskennteleviltä AUKOTON-projektiryhmän jäseniltä. Ideoita työkaluun integroitavasta ohjeistuksesta on esittänyt esimerkiksi Vepsäläinen [14]. Integroidun ohjeistuksen ideointivaiheessa ideoita erilaisista integroitavien ohjeistuksien aiheista oli runsaasti, mutta valitettavasti tätä kirjoitettaessa vain muutamia näistä on saatu toteutettua. AP-työkaluun integroidun ohjeistuksen merkitystä ei pidä silti aliarvioida, koska se on käytännön työssä erittäin tärkeä, ellei jopa tärkein osa ohjeistusta. Voidaan nimittäin pitää hyvin todennäköisenä, että käyttäjät pyrkivät ensisijaisesti löytämään tarvitsemansa ohjeet työkalun yhteydestä. Varsinkin alkuvaiheessa työkalun käyttöön ja kehitysprosessiin etenemiseen käyttäjän tutustuttavat integroidut avusteet ovat osuvia. Vaikka integroitu ohjeistus eroaa toteutukseltaan kirjallisesta ohjeistuksesta, on integroituun ohjeistukseen ammennettu sisältöä kirjallisesta ohjeistuksesta.

6.3. Kirjallinen ohjeistus

Kirjallinen ohjeistus ”User guide for the UML Automation Profile” [30] on asiakirjamuotoinen ohjeistusdokumentti, joka nimensä mukaisesti käsittelee pääasiassa UML-automaatioprofiilia. Automaatioprofiili [28] on spesifikaatio, joka ei pyrikään sellaisenaan olemaan mahdollisimman helposti erilaisten käyttäjien ymmärrettävissä, vaan se pyrkii yksikäsitteisesti kuvaamaan sen sisältämien elementtien tarkoituksen ja yhteydet muihin elementteihin[8]. Automaatioprofiilissa ei siis juuri oteta kantaa elementtien ja kaaviotyyppien käytännön hyödyntämiseen. Kirjallisen ohjeistuksen tarkoituksena onkin ohjata käyttäjää automaatioprofiilin käsitteiden ja kaaviotyyppien oikeanlaiseen käyttöön, eli esitellä hyviä käytäntöjä profiilin ja sen elementtien hyödyntämiseen kehitysprosessin yhteydessä.

Kirjallinen ohjeistus koostuu tätä kirjoitettaessa seuraavista luvuista (mukailtu suomenkielelle ohjeistusdokumentin [30] englanninkielisistä otsikoista): johdanto, UML-automaatioprofiili, Requirements-aliprofiili, AutomationConcepts-aliprofiili, DistributionAndConcurrency-aliprofiili, DevicesAndResources-aliprofiili, automaatioprofiilin kaaviotyypit ja case-esimerkki. Ohjeistus pohjautuu UML-automaatioprofiiliin [28].

[8] Automaatioprofiili ei tältäkään kannalta ole täydellinen.

Ohjeistuksen johdannossa esitellään lyhyesti ohjeistusdokumentin kohderyhmää sekä dokumenttia yleisesti, kuvataan AUKOTON-kehitysprosessia ja esimerkkiprosessi. AUKOTON-kehitysprosessin kuvaus on tässä luvussa yleisellä tasolla eikä sitä sidota vielä kovin voimakkaasti automaatioprofiilin mallinnuselementteihin. Ohjeistusdokumentissa käytettävä esimerkkiprosessi on sama, kuin tämän diplomityön kohdassa 5.2 esitelty esimerkkiprosessi. Esimerkkiprosessilla pyritään yhtenäistämään myöhemmin ohjeistusdokumentissa esiintyvien mallinnuselementtien esimerkkejä niin, että mahdollisimman monen elementin esimerkki olisi esimerkkiprosessin mukainen.

Ohjeistuksen UML-automaatioprofiilia käsittelevässä luvussa käydään läpi profiilin taustaa ja profiilia yleisesti. Käyttäjän kannalta luvun oleellisin osuus lienee kuitenkin profiilin elementtien käytön yleisesittely. Tässä kohdassa käydään läpi se, miten stereotyyppejä käytetään erikoistamaan mallinnuselementtejä ja kuinka tämä heijastuu elementtien ominaisuuksiin (properties). Tämä on siis yksi niistä ohjeistuksen kohdista, joissa UML:än perusteita käsitellään, vaikka ohjeistus ei muuten juuri ota kantaa UML:n perusteisiin.

Seuraavat neljä lukua ohjeistuksessa käsittelevät automaatioprofiilin aliprofiileita pakkauksineen ja elementteineen. Aliprofiileja käsittelevät luvut muodostavat koko ohjeistuksen selkärangan. Luvuissa käydään läpi aliprofiilien pakkaukset ja niiden elementit. Pakkausten ja elementtien käyttöä havainnollistetaan lukijalle automaatioprofiilin spesifikaatiota laajentaen ja tarjoten elementtien käytöstä AP-työkalua käyttäen luotuja graafisia esimerkkejä. Näin pyritään muodostamaan käyttäjälle kuva kunkin elementin graafisesta ilmeestä, käytännön soveltamisesta ja liittymistä muihin mallinnuselementteihin. Katkelma ohjeistusdokumentin mallinnuselementtejä käsittelevästä osasta on liitteessä 1. Katkelmasta käy erityisesti ilmi automaatioprofiilin elementtien käsittelytapa.

Ohjeistuksen kaaviotyyppejä käsittelevässä luvussa käydään läpi automaatioprofiilissa määritellyt kaaviotyypit. Luvussa tarkastellaan kunkin kaaviotyypin käyttötarkoitusta esimerkkien avulla ja automaatioprofiilin tekstuaalista esitystä laajentaen. Lisäksi osion tarkoitus on antaa lukijalle yleiskuva kunkin kaaviotyypin ulkoasusta.

Ohjeistuksen viimeinen varsinainen ohjeistusta käsittelevä luku on case-esimerkki. Tässä luvussa on tarkoitus esitellä yksi tai useampia laajahkoja prosessikokonaisuuksia, jotka case-esimerkissä toteutetaan AUKOTON-kehitysprosessia hyödyntäen. Esimerkkiprosessina on tämän diplomityön kohdassa 5.2 kuvattu prosessi. Toinen mahdollinen mallinnettava esimerkkiprosessi on AUKOTON-projektin keväällä 2009 toteutettu demoprosessi, joka tulee olemaan kohdassa 5.2 esitettyä prosessia monimutkaisempi ja laajempi. Case-esimerkki on ohjeistuksen kannalta tärkeä osa, koska se tarjoaa käyttäjälle kuvan siitä, miten kehitysprosessia hyödyntäen pystytään mallintamaan ja toteuttamaan automaatiosovellus. Lisäksi laajempi esimerkki nivoo yhteen ohjeistuksessa esiteltyjen mallinnuselementtien käyttöä laajemmassa mittakaavassa.

Kirjallisen ohjeistuksen pääasiallinen tarkoitus on antaa automaatiosuunnittelijoille käsikirjamainen kuvaus automaatioprofiilin käsitteiden hyödyntämisestä ja kehitysprosessista esimerkkien avulla, joista suunnittelija voi tarkistaa käsitteistöön ja kehitysprо-

sessiin liittyviä käytäntöjä. Kirjallinen ohjeistus ei kokonaisuudessaan sovi kehitysprosessin ja käsitteistön alkuvaiheen opiskelumateriaaliksi, koska se on varsin pitkä eikä välttämättä muodosta mielekästä kokonaisuutta. Poikkeuksena tähän on ohjeistuksen alkuosa ja esimerkkitapaukset, jotka sopivat opiskelumateriaaliksi varsin hyvin kun suunnittelija tutustuu AUKOTON-kehitysprosessiin ja sen käytäntöihin. Yleisesti ottaen kirjallinen ohjeistus sopii parhaiten syventäväksi ohjeistoksi siinä vaiheessa kun käyttäjä alkaa yksinkertaisten esimerkkitapausten ja AP-työkaluun tutustumisen jälkeen itsenäisesti suunnitella automaatiosovellusta AUKOTON-kehitysprosessin mukaisesti automaatioprofiilin käsitteistöä ja AP-työkalua hyödyntäen.

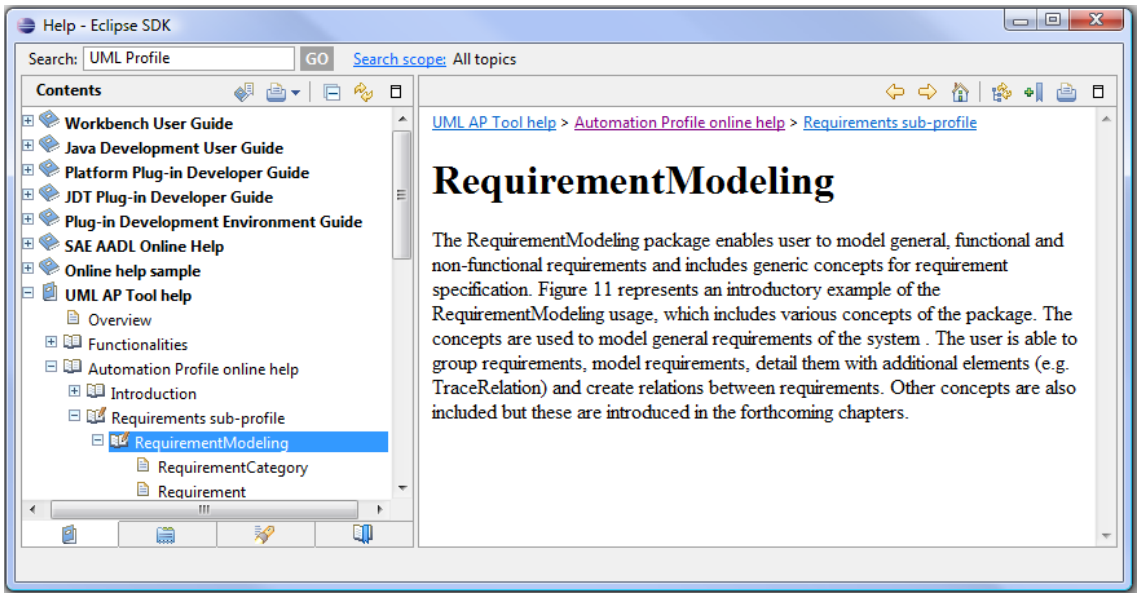
6.4. Työkaluun integroidut ohjeet ja avusteet

Kuten jo edellä kävi ilmi, ohjeistus koostuu kirjallisen ohjeistusdokumentin lisäksi AP-työkaluun integroiduista ohjeista ja avusteista. AP-työkalu on kehitetty Eclipse-työkalualustalle, joka puolestaan mahdollistaa erilaisten ohjeiden ja avusteiden integroimisen osaksi työkalua. Integroidut ohjeet ja avusteet ovat käyttäjän kannalta tärkeitä, koska niiden avulla käyttäjä voi nopeasti päästä käsiksi haluamaansa tietoon esimerkiksi jonkin tietyn mallinnuselementin käytöstä tai kehitysprosessin yleisistä vaiheista.

Tässä aliluvussa esitellään työkaluun integroituja ohjeita ja avusteita. Lisäksi tarkastellaan lyhyesti mitä muita ohjeistus- ja avustemuotoja Eclipse-työkalualusta mahdollistaa.

6.4.1. Työkaluun integroitu kirjallinen ohjeistus

Eclipse-alusta mahdollistaa ohjeiden integroimisen osaksi Eclipsen omaa help-osiota. Help-osio on selattava, erilaisia ohjeita sisältävä dokumenttihierarkia, josta käyttäjä voi valikkohierarkian tai hakujen avulla etsiä ohjeita tai lisätietoja haluamastaan aiheesta. Idealtaan, muttei välttämättä toteutukseltaan, vastaavanlaisia help-osioita löytyy monista muistakin ohjelmistoita kuten Matlab. Myös Windows-käyttöjärjestelmän help on jotakuinkin idealtaan Eclipsen help-osiota vastaava. Kuvassa 6.1 on esitetty näkymä Eclipsen help-osioon. Näkymä jakautuu pystysuunnassa kahteen osaan. Vasemmalla on valikko, joka esittää käyttäjän käytettävissä olevat ohjeiden aiheet. Aiheet avautuvat valikkohierarkiaksi, josta käyttäjä pystyy valitsemaan haluamansa ohjesivun. Oikealla on ikkuna, johon ohjesivut avutuvat. Ikkunan yläreunassa on haku-palkki, jonka avulla käyttäjä voi tehdä sanahakuja ohjetietokantaan.



Kuva 6.1. *Näkymä Eclipse-alustan help-osioon ja esimerkki työkaluun integroidusta ohjesivusta (kuvakaappaus)*

Integroitu ohjeistus sopii erityisesti laajojen kokonaisuuksien ohjeistamiseen työkalun kontekstissa. Myöhemmissä aliluvuissa esiteltävät ohje- ja avustemuodot eivät sovi pitkien tekstuaalisten ohjeistusten esittämiseen. Niistä pystytään kuitenkin usein luomaan linkkejä help-osion sivuille. Help-osion ohjesivuilla on runsaasti tilaa käytettävissä ja mahdollisuus käyttää muun muassa kuvia, joten ohjeista voidaan tehdä tarvittaessa laajoja ja yksityiskohtaisia. Yhtä hyvin voidaan tietenkin tarjota esimerkiksi yksinkertaisia listoja työnkulun vaiheista, mutta tällaisia ohjeita varten Eclipsestä löytyy parempikin avustemekanismi. Diplomityön kontekstissa Eclipsen help-osiota on käytetty kirjallisen ohjeistuksen integroimisessa työkalun yhteyteen. Koko kirjallista ohjeistusta ei ole tätä kirjoitettaessa integroitu työkaluun. Osa kuitenkin on, ja kuvassa 6.1 on esitetty automaatioprofiilin RequirementModeling-pakkauksen AP-työkaluun integroitu ohjeistusteksti Help-osion näkymänä. Toistaiseksi help-osioon on integroitu Requirements-aliprofiilista RequirementModeling- ja AutomationRequirements-pakkausten käsitteistöä.

Eclipsen help-osion laajentaminen omilla ohjehierarkioilla on melko yksinkertaista. Laajentamista varten tulee määritellä XML-muotoinen TOC-tiedosto (Table Of Contents) ja HTML-sivu tai kohde HTML-sivulta kullekin valikkohierarkian kohteelle, jotka on määritelty TOC:issa. TOC-tiedosto kuvaa valikkorakenteen, mutta varsinaiset ohjeet esitetään HTML-muodossa. TOC-tiedostot voivat olla hierarkkisia, eli yksi TOC voi käyttää toista TOC:ia hierarkian määrittelyyn.

Eclipse ajaa help-osiotaan siihen integroidun Apache-palvelimen päällä ja osiota tarkastellaan Eclipseen sisäänrakennetun selaimen avulla. Näin ollen ohjeita sisältävät sivut voivat olla joko HTML- tai esimerkiksi JSP-muotoisia (JavaServer Pages) tai itse asiassa mitä tahansa Apachen tukemaa muotoa. JSP-muotoisilla ohjesivuilla olisi muun muassa mahdollista toteuttaa dynaamisia ohjesivuja. Ohjeet päädyttiin kuitenkin teke-

mään staattisina HTML-sivuina, koska tämä oli ainakin toteutettujen sivujen osalta yksinkertaisempi tehtävä.

Automaatiosuunnittelijoiden kannalta AP-työkalun integroituun help-osioon sijoitettujen ohjeiden anti on suunnilleen vastaava kuin kirjallisen ohjeenkin. Help-osio tarjoaa käyttäjälle syvällisempää tietoa kehitysprosessin ja automaatioprofiilin käsitteistön hyödyntämisestä. Tätä tietoa ei ehkä kokonaisuudessaan kannata hyödyntää käsitteistöön ja kehitysprosessiin tutustuttaessa, vaan se sopii paremmin käyttäjän omaaman tiedon täydennys- ja tarkistusohjeistoksi. Integroidun help-osion paras puoli automaatiosuunnittelijan kannalta on sen helppo saavutettavuus työkalusta, eli toimintaympäristöstä, jossa kehitysprosessia ja käsitteistöä hyödyntävää suunnittelutoimintaa tehdään. Automaatiosuunnittelijan on help-osion ohjeistuksen perusteella helppo tarkistaa haluttu seikka käsitteistöstä tai kehitysprosessista, esimerkiksi millainen informaatiorajapinta tyypillisesti liittyy mittausvaatimukseen.

6.4.2. Cheat sheet

Eclipsen help-osioon integroitavat ohjeet soveltuvat hyvin laajojen ja yksityiskohtaisten kuvausten luomiseen ohjeen aiheesta. Ne eivät kuitenkaan ole kovin tehokkaita, kun käyttäjää halutaan opastaa sovelluksen käyttöön niin, että käyttäjä pääsee itse suorittamaan jotakin ennalta tunnettua työnkulkua. Ongelmaksi muodostuu se, käyttäjä joutuu vuorottelemaan Eclipsen pääikkunan ja help-ikkunan välillä. Tällaisissa tilanteissa parempi käyttäjää avustava mekanismi on cheat sheet. Cheat sheet (CS lyhyesti) on Eclipse-alustan tarjoama mekanismi, joka on tarkoitettu käyttäjän opastamiseen monimutkaisen tehtäväsarjan läpi tietyn tavoitteen saavuttamiseksi. CS on käytännössä pieni näkymä Eclipsen pääikkunassa eikä käyttäjän näin ollen tarvitse vuorotella eri näkymien, esimerkiksi juuri Eclipsen pääikkunan ja help-osion ikkunan, välillä. [43]

CS:ejä käytetään erityisesti erilaisten työnkulkujen kuvaamiseen. Käytännössä CS:it ovat sarja lyhyitä ohjepätkiä, joista kukin kuvaa yhden toimenpiteen suorittamisen kokonaistehtävän kannalta. Ne ovat tehokkaita oppimisen kannalta, koska käyttäjä pääsee itse suorittamaan toimenpiteitä CS:n tarjoamien ohjeiden perusteella. CS antaa kuitenkin mahdollisuuden myös automatisoida toimenpiteiden suorittamista. Käyttäjä voi siis halutessaan tai tilanteen niin vaatiessa antaa CS:n suorittaa kyseisen kohdan tehtävät osin tai kokonaan, esimerkiksi avata dialogin uuden projektin luomista varten. Tämä on tosin mahdollista ainoastaan, mikäli CS:n kehittäjä on tehnyt tämän mahdolliseksi CS:iä toteuttaessaan. Toimintojen automatisoinnin ansiosta CS:it voivat olla hyvinkin interaktiivisia. CS:istä voidaan käynnistää esimerkiksi wizardeja, näkymiä ja muita liitännäisiä. Liitännäisten avulla voidaan, Eclipsen liitännäismekanismin puitteissa, toteuttaa lähes mitä tahansa toiminnallisuutta, joten monimutkaistenkin toimintojen automatisointi tai käynnistys on mahdollista.

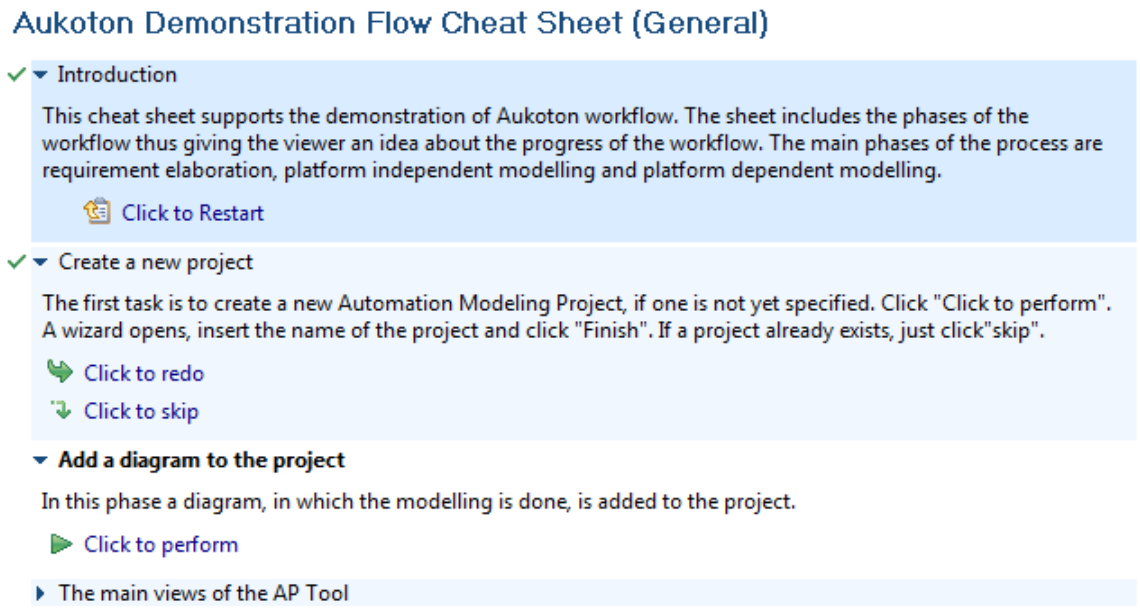
Toinen tärkeä CS:ien ominaisuus käyttäjän kannalta on mahdollisuus linkittää työvaiheiden suppeita kuvauksia edellisessä aliluvussa esiteltyyn Eclipsen help-osioon. CS:ien ohjeet on tarkoitus pitää suhteellisen lyhyinä, muutamien virkkeiden mittaisina ”pätkinä”, joiden perusteella käyttäjä kuitenkin pystyy suorittamaan halutun toimenpi-

teen. Käytännössä ohjeet kuitenkin venyvät paljon pidemmiksi. Laajempi ja yksityiskohtaisempi kuvaus suoritettavasta toimenpiteestä, sen esiehdoista tai esimerkiksi lopputuloksista voidaankin esittää Help-osiossa, johon voidaan suoraan linkittää CS:in toimenpidettä käsittelevästä kohdasta. Näin käyttäjä pystyy helposti saamaan lisätietoja niissä CS:in kohdissa, joissa hän sitä tuntee tarvitsevansa.

CS:it ovat liitännäisiä, jotka laajentavat Eclipseä. CS:it toteutetaan käytännössä kahden XML-muotoisen tiedoston avulla. Ensimmäinen tiedostoista, plugin.xml, kuvaa muun muassa sen, miten CS liittyy Eclipseen, eli miten CS laajentaa sitä. CS:ien tapauksessa plugin.xml määrittelee muun muassa Eclipsen laajennuspisteeksi cheat sheet content pisteen. Lisäksi plugin.xml tiedostossa määritellään se mihin kategoriaan kukin liitännäisen CS:eistä kuuluu, eli miten määriteltävä CS asettuu Eclipsestä löytyvään CS-valikkoon, johon on koottuna kaikki kohteena olevan Eclipsen asennukseen kuuluvat CS:it. Toinen tarvittavista tiedostoista on myös XML-muotoinen ja se kuvaa CS:in rakenteen ja sisällön. Kutsutaan sitä tässä CS-tiedostoksi. CS-tiedosto kuvaa CS:in koko sisällön, muun muassa määrittelemänsä CS:in yksittäiset tehtävät, niiden tekstisisällön, mahdolliset linkit help-osioon ja käyttäjän puolesta suoritettavat toiminnot. CS-tiedoston sisältö voidaan luoda joko käsin sopivaa XML-notaatiota kirjoittaen tai käyttämällä Eclipsestä löytyvää yksinkertaista editoria, jolla sopivan muotoisia CS-tiedostoja voidaan rakentaa ilman XML-notaation tuntemusta.

CS:it voidaan jakaa kahteen ryhmään: yksinkertaiset ja yhdistelmä-CS:it. Yksinkertaiset CS:it ovat nimensä mukaisesti CS:ien perustoiminnot omaava muoto, joita kannattajaa hyödyntää lyhyehköissä työkulkujen kuvauksissa, jotka muodostavat yhden kokonaisuuden. Yhdistelmä-CS:it mahdollistavat yksinkertaista versiota monipuolisemman toiminnallisuuden. Ne koostuvat yksinkertaisista CS:eistä joiden suoritusta voidaan kontrolloida varsin tarkasti. Yhdistelmä CS:iä kannattaa hyödyntää laajoissa työkulkujen kuvauksissa kuten esimerkiksi laajojen kehitysprosessien ohjeistamisessa, jotka koostuvat useista itsenäisistä kokonaisuuksista.

Kuvassa 6.2 on esitetty osa (yksinkertaisesta) CS:istä, joka opastaa käyttäjän läpi AUKOTON-kehitysprosessin (itse asiassa tässä tapauksessa kyse on paremminkin kehitysprosessia yleisesti esittelevästä CS:istä). Tässä esitetty CS on korkealla abstraktiotasolla, eikä pelkästään sen perusteella ole tarkoituskaan kyetä suoriutumaan kehitysprosessin läpikäynnistä ilman tarkempaa ohjeistusta. Paremminkin kyseessä on muistilistaa muistuttava CS, joka toimii käyttäjän tukena pitkän kehitysprosessin suorittamisessa. Kyseinen CS on AUKOTON-projektin ensimmäistä demotilaisuutta varten toteutettu demonstraatio CS:in toiminnasta. Kuvan tilanteessa käyttäjä on suorittanut kaksi ensimmäistä kohtaa CS:istä. Käyttäjän tulisi CS:in mukaan seuraavaksi luoda uusi kaavio projektiin. CS avaa kaavion luontiin tarkoitetun wizardin kun käyttäjä painaa ”Click to perform”-valintaa. Tässä tapauksessa käyttäjälle ei ole annettu ohjeita toimenpiteen suorittamiseen, joten automatisoidun toiminnon käyttö on suositeltavaa. CS on esitetty kokonaisuudessaan Liitteessä 2.



Kuva 6.2. *Osa cheat sheetistä, joka opastaa käyttäjää AUKOTON-kehitysprosessin käyttöön*

Tätä kirjoitettaessa AP-työkaluun ei ole vielä implementoitu useita CS:ejä, vaikka potentiaalisia CS-ehdokkaita on runsaasti. Muutama toteutettu CS AP-työkalulle ja AUKOTON-kehitysprosessille on kuitenkin olemassa, joista yksi on osittain esitetty kuvassa 6.2. On kuitenkin todettava, että CS sopii olemukseltaan parhaiten AP-työkalun käyttöä opastaviin tehtäviin, jotka eivät ole diplomityön varsinaista ydinaluetta.

CS on merkityksellinen automaatiosuunnittelijalle hänen tutustuessaan AP-työkaluun ja kehitysprosessiin. Eclipse on automaatiosuunnittelussa varsin vähän hyödynnetty ympäristö. Tästä syystä CS:ien parasta antia automaatiosuunnittelijoille ovat työkalun ja ympäristön käyttöön liittyvät yksityiskohtaiset ja vaiheistetut työnkulut, joiden avulla suunnittelijat pääsevät sinuiksi työkalun ja ympäristön kanssa.

6.4.3. Wizard

Työkaluun integroituun ohjeistukseen ja cheat sheet:eihin verrattuna toisenlaista näkökulmaa käyttäjän avustamiseen tuo Eclipsen tarjoama wizard-mekanismi. Wizardia (velho) sovelletaan yleensä tilanteissa, joissa jotakin rutiininomaista tai monimutkaista tehtävää halutaan automatisoida. Tällainen tehtävä on esimerkiksi uuden projektin luominen, jonka suorittamiseksi käyttäjän pitäisi manuaalisesti määritellä lukuisia asetuksia ja tiedostoja. Tämänkaltaisen toiminnon wizard pystyy muutamien käyttäjältä saatavien tietojen perusteella toteuttamaan sekunneissa.

Wizardeja hyödyntämällä voidaan muun muassa saavuttaa toiminnan tehostamista ja toimintojen lopputulosten laadun paranemista. Toiminnan tehostamiseen päästään sellaisissa käyttökohteissa, joissa wizard pystyy tuottamaan halutun lopputuloksen käyttä-

jältä pyydettyjen tietojen pohjalta ja yhdistää nämä tiedot käyttämäänsä malliin[9]. Malli, jota wizard käyttää, voidaan rakentaa wizardin oman sovelluskoodin sisään tai toteuttaa omana luokkanaan, joista viimeksi mainittu on yleensä elegantimpi ratkaisu. Yksinkertaisessa tapauksessa wizardin sisään rakennettu malli on kuitenkin täysin riittävä, joskin hankalammin ylläpidettävä ratkaisu. Wizard käyttää mallia kooditasolla ja tuottaa näin käyttäjän syötteiden mukaisen lopputuloksen. Lopputuloksen laatua pystytään nostamaan, koska wizard hyödyntää mallia lopputuloksen generointiin. Kun wizardin käyttämä malli on saatu toimimaan oikein, ovat wizardin tuottamat lopputulokset valideja. Tämän ehtona on kuitenkin se, että wizard estää käyttäjää syöttämästä sellaisia lähtötietoja, jotka eivät tuota validia lopputulosta.

Havainnollistetaan mainittuja hyötynäkökohtia esimerkillä, jossa käyttäjän tulee luoda uusi projekti AP-työkaluun. Toiminnan tehostuminen saavutetaan helposti, koska projektin luominen manuaalisesti Eclipse-alustalle on kohtuullisen raskas tehtävä, jossa käyttäjän tulee itse määritellä tarvittavat kansio- ja pakettirakenteet sekä projektin asetukset. Wizardin avulla tehtävän pystyy suorittamaan nopeasti, koska tällöin käyttäjän tulee syöttää ainoastaan luotavan projektin nimi. Lopputuloksen laatu paranee wizardia käytettäessä osin samoista syistä kuin toiminto nopeutuu. Koska manuaalisessa projektin luomisessa käyttäjä joutuu määrittelemään paljon tietoa käsin, on mahdollisuus inhimillisille virheille olemassa. Täten mahdollisuus piileville virheille on suurempi kuin malliin pohjautuvassa lopputulosten generoinnissa, olettaen että wizardin käyttämä malli on huolellisesti toteutettu ja testattu.

Wizardit koostuvat mielivaltaisesta määrästä sivuja, joilla käyttäjältä kerätään tietoja lopputulosten generointia varten. Wizardien toteutuksessa periaatteena on, että käyttäjä voi liikkua vapaasti sivuilla ja täyttää tarvittavat tiedot haluamassaan järjestyksessä. Toinen tärkeä periaate on se, että käyttäjä pystyy missä tahansa vaiheessa lopettamaan tietojen syöttämisen ja generoimaan lopputulokset. Käytännössä tämä tarkoittaa sitä, että wizardien toteutuksessa pitää mahdollisimman monille syöteille antaa mielekäs oletusarvo, jota voidaan käyttää siinä tapauksessa, että käyttäjä ei antanut kyseistä tietoa. Aina tämä ei ole mahdollista, mutta yleensä voidaan tehdä jokin oletus sopivasta arvosta esimerkiksi sen perusteella, kumpaa kahdesta mahdollisesta vaihtoehdosta pidetään yleisempänä. Tärkeä osa wizardia on myös käyttäjän syötteiden mielekkyyden tarkistaminen. Wizardin ei tule sallia käyttäjältä sellaista syötettä, joka on sen toiminnan kannalta sopimaton. Tästä syystä wizard yleensä lukitsee itsensä, eli ei anna käyttäjän vaihtaa sivua tai generoida lopputuloksia jos laiton syöte havaitaan syötekentässä.

Eclipse-kehitysalusta sisältää sisäänrakennetun mekanismin wizardien tekemiseen. Käytännössä tämä tarkoittaa sitä, että Eclipse tarjoaa valmiit luokat, joilla wizard voidaan helpohkosti toteuttaa. Wizardin tekemiseen tarvittavat Eclipsen tarjoamat luokat ovat Wizard- ja WizardPage-luokat, joita käyttäjä voi halutessaan laajentaa. Wizard-luokka tarjoaa metodit muun muassa wizardin alustamiseen, sivujen lisäämiseen ja sivu-

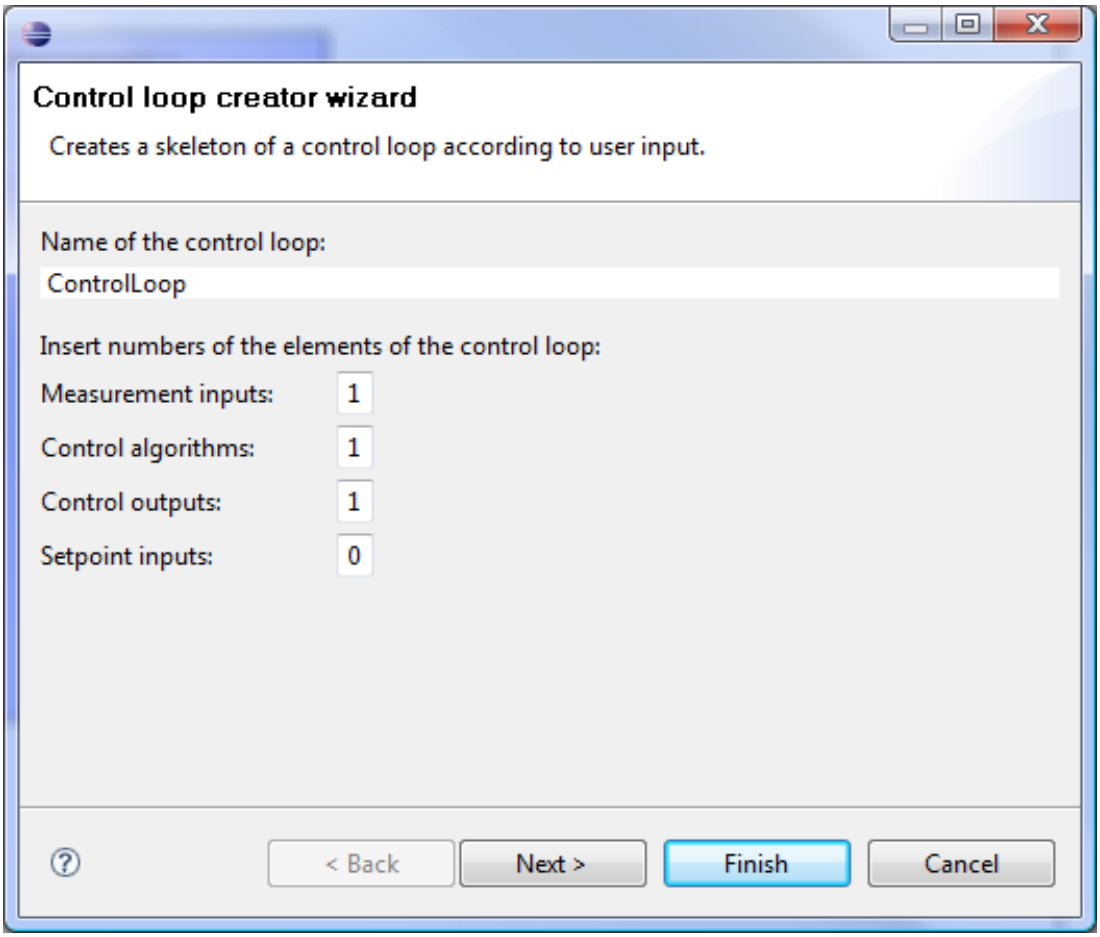
[9] Tässä yhteydessä mallilla tarkoitetaan sitä perustaa, jonka mukaan wizard rakentaa lopputulokset. Wizardissa malli siis kuvaa sitä toimintatapaa miten käyttäjän antamat syötteet kuvautuvat generoitavaan lopputulokseen.

jen selaamisen kontrollointiin. WizardPage puolestaan on luokka, jolla määritellään wizardiin liitettävien sivujen rakenne ja toiminnallisuus. WizardPage-luokka tarjoaa muun muassa metodit sivun rakenteen sekä muokkaus- ja sivunvaihtotapahtumien määrittelyyn. Itse wizard siis muodostuu Wizard-luokan oliosta joka puolestaan koostuu WizardPage-luokan olioista. Wizardin rakentajassa määritellään siinä käytettävät sivut (WizardPage), jotka lisätään wizardiin.

Eclipsen wizard-järjestelmä ei tue dynaamista sivujen poistamista, joten dynaamisten wizardien toteuttaminen on hankalaa. Sivumäärältään dynaamisen wizardin tekeminen on toki mahdollista, mutta se on varsin haastavaa, koska tämä vaatii käytännössä oman toteutuksen wizardin sivunhallintajärjestelmään. Tästä syystä tämän diplomityön puitteissa tehty wizard on sivumäärältään staattinen, eli wizardin sivut määritellään kiinteästi, jolloin monimutkaiselta sivunhallinnan toteutukselta vältytään. Lisäksi kiinteä sivumäärä helpottaa wizardin käyttöä, koska se on käyttökerrasta toiseen samanlainen.

AP-työkaluun on sen kehittäjän toimesta toteutettu muutamia normaalissa käytössä tarpeellisia wizardeja, kuten edellä mainitussa esimerkissä esitetty uuden projektin luova wizard. Tämän diplomityön puitteissa työkaluun toteutettiin enemmän käyttäjää avustava wizard, joka ei varsinaisesti ole työkalun käytön kannalta välttämätön, vaan nimenomaan käyttäjää mallinnustehtävän suorittamisessa avustava. Diplomityön puitteissa toteutettu wizard luo käyttäjän antamien tietojen perusteella käsiteltävään malliin säätösilmukan. Säätösilmukan luominen käsin on aikaa vievä tehtävä. Tämä johtuu lähinnä AP-työkalun nykyisestä tavasta käsitellä elementtien lisäämistä malliin, missä kukin elementti ja portti on lisättävä malliin erikseen. Keskimääräinen säätösilmukka koostuu ainakin neljästä peruselementistä (ControlLoop, MeasurementInput, ControlAlgorithm ja ControlOutput), joihin lisäksi liittyy keskimäärin muutama portti elementtiä kohti. Näin ollen yksinkertaisenkin säätösilmukan toteuttaminen manuaalisesti on varsin raskas operaation käyttäjälle. Toteutetulla wizardilla halutaankin helpottaa käyttäjän taakkaa säätösilmukkaa luotaessa.

Toteutettu wizard (Control loop creator) luo käyttäjän syötteiden perusteella halutun määrän kutakin säätösilmukan peruselementtiä alustaen ne käyttäjän haluamalla stereotyypillä sekä elementeille tyypillisillä porteilla, jotka on myös alustettu sopiviin oletusarvoihin. Kun käyttäjä on syöttänyt haluamansa tiedot, wizard luo elementit malliin ja tämän jälkeen piirtää ne diagrammiin sopivan kokoisina.



Kuva 6.3. *Control loop creator wizardin aloitussivu*

Kuvassa 6.3 on esitetty toteutetun Control loop creator wizardin aloitussivu. Aloitussivulla käyttäjä valitsee luotavan säätösilmukan nimen ja silmukkaan liitettävien elementtien lukumäärät. Wizardin kirjoitushetken toteutuksessa käyttäjä voi valita mittaussisääntulojen, säätöalgoritmien, ohjauslähtöjen ja asetusarvotulojen lukumäärät. Kun käyttäjä on antanut lukumäärät, voi käyttäjä halutessaan lopettaa tietojen syöttämisen ja generoida säätösilmukan. Käyttäjä voi kuitenkin jatkaa tietojen syöttämistä seuraavilla sivuilla, joilla käyttäjä voi valita elementeille tarkemmat stereotyypit wizardin tarjoamasta listasta ja valita alustetaanko elementit porteilla vai ei. Stereotyyppejä voi lisätä MeasurementInput, ControlAlgorithm ja ControlOutput elementeille. Lisäksi käyttäjä voi lisätä luotavaan säätösilmukkaan lukituselementtejä. Control loop creator wizard on toteutettu edellä mainittujen wizardien toteutusperiaatteiden mukaisesti, eli se sallii vapaan sivujen selailun, se tarkistaa käyttäjän syötteet ja lukkiutuu virheellisen syötteen saadessaan sekä mahdollistaa tulosten generoinnin oletusarvoilla alusta asti. Esimerkkitapaus, jossa esitetään kaikki wizardin sivut ja jossa käydään läpi eräs mahdollinen wizardin käyttötapaus, on löydettävissä liitteestä 3.

Wizardit eivät automaatiosuunnittelijan näkökulmasta tarjoa kovinkaan paljon varsinaista ohjeistuksellista arvoa. Automaatiosuunnittelija ei opi wizardia käyttämällä mitään kehitysprosessista tai sen käsitteistöstä. Ainakaan wizardeja ei ole suunniteltu tätä tarkoitusta silmälläpitäen. Wizardien anti automaatiosuunnittelijalle onkin avusta-

van ja käyttöä tehostavan toiminnallisuuden tarjoaminen suunnittelutyöhön. Nämä ominaisuudet korostuvat kun automaatiosuunnittelija tuntee kehitysprosessin ja osaa hyödyntää sen käsitteistöä. Tällöin käyttäjä pystyy wizardien avulla tehostamaan suunnittelutyötänsä.

6.4.4. Muita työkalualustan mahdollistamia ohjeistus- ja avustemuotoja

Edellä esitellyt Eclipseen integroitavat ohjeistukset ja avusteet eivät ole ainoita käyttäjää ohjeistavia ja avustavia toimintoja, joiden käytön Eclipse mahdollistaa. Tällaisiksi voidaan laskea muun muassa tooltipit. Tooltipilla (työkaluvihje) tarkoitetaan yleensä lyhyttä tekstiä, joka ilmestyy näkyviin pidettäessä osoitinta hetki paikallaan objektin päällä, jolle tooltip on määritelty. Tooltip voi sisältää esimerkiksi objektin määrittelemän toiminnon näppäinoikotien. Tämänkaltaisella toiminnallisuudella voisi olla käyttöä AP-työkaluun integroitavan ohjeistuksen toteuttamisessa, mutta sen tarjoamaa hyötyä voidaan tässä vaiheessa pitää varsin vähäisenä.

Työkaluun ei diplomityön puitteissa ole juurikaan kehitetty tooltipejä. Edellisessä aliluvussa esiteltyyn wizardiin on testimielessä lisätty muutamia tooltipejä sellaisiin syötekenttiin, joissa käyttäjälle halutaan tarjota selvennystä syötetiedon tarkoituksesta ja muodosta. Tooltip on varsin yksinkertainen avustemekanismi. Se on käytännössä tekstikuvaus, joka liitetään siihen graafiseen elementtiin, jolle tooltip halutaan mahdollistaa. Tyypillinen ratkaisu on upottaa kuvausteksti suoraan lähdekoodiin, joka on useissa tapauksissa sopiva ratkaisu. Hieman elegantimpi tapa olisi käyttää tietokantaratkaisua, joko AP-työkalun lähdekoodiin upotettua tai erillistä tietokantaa, josta tooltipit haettaisiin latauksen yhteydessä ja jossa niitä voitaisiin keskitetysti hallitta.

7. JOHTOPÄÄTÖKSET

Tässä luvussa pohditaan ohjeistuksen onnistumista eri tahojen arvioiden kautta ja esitetään jatkokehitysideoita ohjeistukseen liittyen. Arvioinnit koostuvat tekijän omista, AUKOTON-projektityöryhmän jäsenten ja projektissa mukana olleiden yritysten edustajien arvioista. Arvioissa esitetään näkemyksiä ohjeistuksen ja sen toteutuksen onnistumisesta eri näkökulmista, muun muassa tarkastellen ohjeistuksen soveltumista käyttötarkoitukseensa sekä sen yleistä laatutasoa. Jatkokehitysideoihin on koottu eri tahojen näkemyksiä ohjeistuksen jatkokehitysmahdollisuuksista ja -tarpeista.

7.1. Tekijän näkemyksiä ohjeistuksesta ja sen toteutuksesta

Tähän asti toteutettua ohjeistusta voidaan pitää varsin onnistuneena kokonaisuudessaan, mutta muutamia ongelmakohtia on kuitenkin nostettava esiin. Ohjeistuksen tekoa aloitettaessa ei tekijällä ollut kattavaa kokonaiskuvaa kehitysprosessin etenemisestä[10]. Kehitysprosessi nähtiin varsin epämääräisenä, kunnes joulukuussa 2008 pidetty AUKOTON-projektin demonstraatio selkiytti kehitysprosessin kulun. Tästä syystä varsinkin kirjallisen ohjeistuksen kaksi ensimmäistä automaatioprofiilin aliprofiileja käsittelevistä luvuista on hieman heikosti sidottu kehitysprosessiin. Toisaalta myös kahdessa jälkimmäisessä aliprofiileja käsittelevässä luvussa sitominen kehitysprosessiin on jäänyt hieman löyhäksi. Näiden lukujen osalta syynä voidaan pitää sitä, että toistaiseksi viimeksi mainittuja automaatioprofiilin laitteistoläheisiä aliprofiileja ei ole juurikaan sovellettu varsinaisessa kehitysprosessissa, vaan AUKOTON-ketjussa on keskitytty lähinnä vaatimusten ja automaatiotoimintojen mallintamiseen. Näin ollen ohjeistuksessa ei riittävän selkeästi kuvata sitä, missä kehitysprosessin vaiheessa kyseisiä mallinnuskäsitteitä olisi tarkoitus käyttää. Toinen seikka mihin ohjeistuksen suhteen ei voida olla täysin tyytyväisiä, on AP-työkaluun integroidun ohjeistuksen määrä. Kirjallisen ohjeistuksen saattaminen siihen tilaan, jossa se tätä kirjoitettaessa on, vei paljon enemmän aikaa, kuin alun perin arvioitiin. Tämä on suurin syy integroidun ohjeistuksen suhteellisen pieneen määrän.

Ohjeistuksessa on kuitenkin nähtävissä myös hyvin onnistuneita kohtia. Varsinkin automaatioprofiilia käsittelevän kirjallisen ohjeistuksen eduksi voidaan lukea kattava esimerkkien käyttö. Lähes jokaiselle profiilin elementille, jota voidaan mallintamisessa käyttää, löytyy ohjeistuksesta graafinen esimerkki. Tätä voidaan pitää ohjeistuksen suurimpana vahvuutena automaatioprofiiliin nähden. Automaatioprofiili kyllä määrittelee kaikki elementit kohtuullisesti, mutta se ei tarjoa juurikaan tietoa niiden käytännön so-

[10] Tässä vaiheessa kehitysprosessi oli luultavasti ainakin hieman epämääräinen myös muille AUKOTON-projektityöryhmän jäsenille.

veltamisesta. Tästä syystä ohjeistuksen antia sen puutteista ja ongelmista huolimatta voidaan pitää AUKOTON-projektille tärkeänä, koska se tarjoaa käytännön opastusta automaatioprofiilin ja kehitysprosessin soveltamisesta.

Ohjeistuksen voidaan katsoa tarjoavan automaatiosuunnittelijalle lisäarvoa automaatioprofiiliin ja AP-työkalulle sen käyttöohjeisiin nähden. Ajateltaessa asiaa automaatiosuunnittelijan kannalta, joka ei kovin hyvin tunne UML:ää ja mallipohjaisia menetelmiä, voidaan nähdä, että ohjeistus jo nykymuodossaan tarjoaisi apua suunnitteluprosessin ja siinä sovellettavan käsitteistön käyttöön. Voidaan pitää myös täysin mahdollisena, että ohjeistuksen ja kohtuullisen vähäisen koulutuksen avulla suunnittelijat pystyisivät suorittamaan yksinkertaisia mallinnustehtäviä AUKOTON-kehitysprosesia käyttäen.

Koska ohjeistuksen saattaminen nykyiseen tilaansa on ollut varsin suuri urakka, on syytä arvioida myös sitä, miten ohjeistus on toteutettu ja miten siinä on onnistuttu. Jälkikäteen voidaan todeta, että lähestymistapa ohjeistuksen toteuttamiseen ei ollut oikeastaan missään suhteessa optimaalinen. Ohjeistusta lähdettiin tekemään vailla todenmukaista käsitystä automaatiosuunnittelijoiden tarpeista ohjeistuksen suhteen. Tietenkin jonkinlainen peruskäsitys ohjeistuksen tarvittavasta sisällöstä oli olemassa, mutta oikea tapa olisi kuitenkin ollut haastatella automaatiosuunnittelijoita ja kysyä heiltä, minkälaista ohjeistusta he haluaisivat ja tarvitsisivat pystyäkseen soveltamaan käsitteistöä ja kehitysprosessia käytännössä. Tämä olisi toisaalta vienyt paljon aikaa, eikä ohjeistusta olisi valmiina niin paljon kuin sitä nyt on. Toisekseen tiedot kehitysprosessista ja automaatioprofiilista olivat sen verran heikot työn alkuvaiheessa, että niiden perusteella ei välttämättä olisi edes pystytty kysymään oikeita asioita. Tässä vaiheessa voidaan kuitenkin todeta, että seuraavalla kerralla valmiudet ohjeistuksen toteuttamiselle ovat merkittävästi paremmat, koska kohderyhmän tarpeiden tunteminen on tärkeää ohjeistuksen toteuttamisessa. Eräs mahdollisuus olisi ollut toteuttaa ohjeistuksesta ensin pieni osa ja testata sitä kohderyhmällä.

Toinen tärkeä seikka, jossa ohjeistuksen toteutuksessa osin epäonnistuttiin, oli ohjeistuksen toteutusformaatti. Ohjeistus kirjoitettiin Microsoft Word 2007 tekstinkäsittelyohjelmalla asiakirjamuotoon. Asiakirja on varsin hyvä ohjeistusformaatti, mikäli se tulee olemaan myös ainoa ohjeistusformaatti, jota tullaan käyttämään. Kuitenkin jo ohjeistuksen toteutuksen alkuvaiheilla oli tiedossa, että ohjeistusta tullaan hyödyntämään myös muissa muodoissa, esimerkiksi HTML-muodossa. Asiakirjan muuttaminen HTML-muotoon käsin on varsin suoraviivainen prosessi, joskin aikaa vievä. Varsinainen ongelma tuleekin vastaan siinä vaiheessa kun HTML-ohjeistus on kertaalleen toteutettu ja ohjeistusta muokataan. Tällöin joudutaan myös HTML-muotoinen ohjeistus korjaamaan käsin niiltä osin, joihin muokkauksia on tehty. Tätä asiaa ei ohjeistusta alun perin toteutettaessa otettu mitenkään huomioon. Word 2007 tukee dokumentin tallentamista HTML-muotoon, mutta sen tuottama HTML-dokumentti ei ainakaan sellaisenaan sovi Eclipsen help-osiossa käytettäväksi niin, että Eclipsen puolella olisi helppoa määritellä kutakin ohjeen aihetta vastaava HTML-dokumentti tai HTML-dokumentin osa.

Diplomityön loppuvaiheessa yritettiin etsiä suoraviivaista tapaa muuttaa suuri asiakirjadokumentti Eclipsen help-osion ymmärtämään HTML-muotoon, jossa jokaista ai-

hetta (automaatioprofiilin elementtiä) vastaa yksi HTML-sivu. Tällaista tapaa ei kirjoitushetkeen mennessä ole löydetty. Toisenlainen ratkaisu ongelmaan kuitenkin löydettiin. Suosituksena vastaaviin tilanteisiin voidaankin todeta, että asiakirjadokumentin sijaan ohjeistukset, joita aiotaan hyödyntää erilaisissa muodoissa, kannattaa toteuttaa tarkoitusta varten olemassa olevilla ohjeistustyökaluilla, kuten Adobe RoboHelp [44]. Nämä työkalut tukevat ohjeistuksen toteuttamista yhteisen lähteen pohjalta (single source publishing), jossa ohjeistus toteutetaan työkalussa ja lopulliset ohjeistukset generoidaan toteutuksen perusteella haluttuun muotoon. Työkalut tukevat yleensä ainakin yleisimpiä ohjeistusmuotoja, kuten HTML-pohjaista ohjeistusta.

7.2. AUKOTON-projektityöryhmän jäsenten arvioita ohjeistuksesta

Ohjeistuksesta pyydettiin arvioita ja palautetta tätä diplomityön osaa varten myös muilta AUKOTON-projektityöryhmän jäseniltä. Näin pystytään antamaan kuva myös siitä, kuinka hyvin ohjeistus on heidän mielestään onnistunut eri osa-alueilla. Kysely koski kirjallista ohjeistusta ja lähinnä sen automaatioprofiilia käsittelevää osaa. Kysely suoritettiin pyytämällä projektityöryhmän jäseniä lukemaan osa ohjeistuksesta sekä vastaavat osat automaatioprofiilista ja vastaamaan lomakkeella lähetettyihin ohjeistusta koskeviin kysymyksiin. Kyselyyn tuli muutamia vastauksia, mikä on tässä tapauksessa riittävä määrä, koska tarkoituksena ei ollut tehdä tilastollista tutkimusta, vaan kerätä näkemyksiä ja kehitysehdotuksia ohjeistukseen liittyen. Kyselyssä ilmenneisiin ohjeistuksen kehityskohteisiin palataan tarkemmin seuraavassa aliluvussa.

Kyselyn päätarkoituksina oli selvittää kuinka hyvin ohjeistus palvelee automaatiosuunnittelijaa, kuinka ymmärrettävä ohjeistus on ja tarjoaako ohjeistus lisäarvoa automaatioprofiiliin nähden? Tässä ja seuraavissa kappaleissa tarkastellaan arvioijien näkemyksiä ohjeistuksesta mainittujen kysymysten näkökulmasta unohtamatta muita kyselyssä esiin tulleita seikkoja. Arvioijien mielestä ohjeistus oli varsin ymmärrettävä, varsinkin sellaisille henkilöille, joilla on edes jonkinlainen käsitys UML:stä. Vastauksista näkyi kuitenkin se, että UML-kieltä heikosti tuntevalle henkilölle ohjeistus ei välttämättä tarjoa riittävästi opastusta. Toinen seikka, joka vastaajien mielestä haittasi ohjeistuksen ymmärrettävyyttä oli sen kieliasu, jossa oli heidän mukaansa parantamisen varaa. Eräs arvioijista kuitenkin mainitsi kieliasun parantuneen ohjeistuksen loppua kohti, joten ohjeistuksen kirjoittaminen on näin ollen ainakin hieman parantanut tekijän englannin kielen taitoja, mikä on tietenkin erinomainen asia.

Eräs kyselyn tavoitteista oli selvittää, kuinka paljon ohjeistus tarjoaa lisäarvoa suunnittelijalle automaatioprofiilin nähden. Arvioijien mukaan ohjeistus tuo lisäarvoa profiilispesifikaatioon nähden esimerkiksi AP-työkaluun sidottujen esimerkkien ja suunnittelijalähtöisesti kuvattujen mallinnuselementtien esittelyn ansiosta. Toisaalta esiin nousi se seikka, että ohjeistus muistuttaa kovasti automaatioprofiilia ja jopa toistaa sitä. Tämä seikka nousi esiin jo profiilin ohjeistusta kirjoitettaessa, mutta automaatioprofiili oli

kuitenkin käytännössä ainoa lähde, jota pystyttiin hyödyntämään ohjeistuksen toteutuksessa automaatioprofiilin elementtien osalta.

Seikka, joka lopulta ratkaisee ohjeistuksen onnistumisen, on sen soveltuminen kohderyhmänsä, eli automaatiosuunnittelijoiden (ja opiskelijoiden), käyttöön. Arvioijien mukaan ohjeistus saavuttaa tämän tavoitteen varsin hyvin. Ohjeistusta pidettiin hyödyllisenä varsinkin kehitysprosessin ja käsitteistön opiskelun alkuvaiheessa. Toisaalta ohjeistus sai osakseen kritiikkiä muun muassa UML:n jäämisestä etäiseksi sekä suunnitteluprosessin hyvin yleisen tason esittelystä. Arvioijien mielestä ohjeistus tarjosi melko hyvin informaatiota automaatioprofiilin käyttämiseen ja tulkintaan, mutta näiden osalta tarvittaisiin heidän mielestään kuitenkin tarkennuksia ja lisää käytännön esimerkkejä. Ohjeistuksen rakennetta pidettiin varsin onnistuneena, mutta tiivistämistä ja lyhennettyä versiota toivottiin.

Projektityöryhmältä ohjeistuksesta saatu palaute oli yleisesti ottaen varsin positiivista, mutta jokaisessa arviossa esiintyi myös asiaankuuluvaa kritiikkiä ja kehitysehdotuksia, joten paljon on vielä parannettavaa. Ohjeistuksen toteuttajan kannalta tämä on tietenkin mukava asia, koska positiivinen yleiskuva kertoo siitä, että jotain on tehty oikein ja kehitysideat ja kritiikki taas auttavat osaltaan kehittämään ohjeistusta oikeaan suuntaan.

7.3. Automaatiosuunnittelijoiden arvioita ohjeistuksesta arviointitapahtuman pohjalta

Ohjeistus on alun perin suunnattu automaatiosuunnittelijoille, joten heidän mielipiteensä ja näkemyksensä ohjeistuksesta ovat erityisen tärkeitä ohjeistuksen arvioinnissa. Automaatiosuunnittelijoiden arvioita ohjeistuksesta kerättiin AUKOTON-projektin arviointitapahtumassa[12]. Arviointitapahtuma järjestettiin syksyllä 2009 ja sen tarkoituksena oli kerätä projektin yritysedustajilta empiriaa projektin tulosten arviointia varten. Tavoitteena oli kerätä tietoa AUKOTON-projektin eri osa-alueilta kuten kehitysprosessista, AP-työkalun käytöstä ja käytettävyydestä, ohjeistuksesta sekä mallinnuskäsitteistöstä (automaatioprofiili). Arviointitapahtumassa yritysedustajat toteuttivat yksinkertaisen automaatiosovelluksen, joka ei kuitenkaan edustanut triviaalia yhden säätöpiirin järjestelmää, vaan kehitettävä sovellus koostui useista säätöpiireistä ja sisälsi muun muassa kaskadisäätörakenteen. Yritysedustajat suunnittelivat sovelluksen annettujen lähtötietojen, kuvausten ja ohjeiden mukaisesti. Periaatteena oli, että kutakin vaihetta kohden oli yksi hyvin ohjeistettu suunnittelutehtävä ja toisessa suunnittelutehtävässä osallistujat joutuivat soveltamaan ensimmäistä tehtävää varten annettuja tarkkoja ohjeita.

Tapahtumassa käyttökokemusten, arvioiden ja mielipiteiden kerääminen suoritettiin haastattelujen ja havainnoinnin perusteella. Osallistujia havainnointiin ja heidän kohtaamistaan ongelmista tehtiin muistiinpanot, jotta kyseisiin ongelmakohtiin voitaisiin

[12] Vaikka arviointitapahtumaan yhteistyöyrityksistä osallistuneista henkilöistä kukaan ei ollut varsinaisesti automaatiosuunnittelija, voidaan heidän katsoa olevan ohjeistuksen arvioinnin näkökulmasta automaatiosuunnittelijoita.

puuttua jatkokehitysvaiheissa. Lisäksi osallistujia haastateltiin kehitysprosessin eri vaiheiden välillä. Haastattelujen ja ongelmakirjausten avulla pystyttiin tekemään yhteenvetoa siitä, missä on onnistuttu ja missä on vielä jatkokehitettävää.

Ohjeistuksesta kerättävien arviointien osalta tarkoituksena oli, että suunnittelijoiden näkemyksiä ohjeistuksesta kerättäisiin pääasiassa heille työnkulun aikana aiheutuneiden ongelmatilanteiden yhteydessä. Suunnittelijoita pyydettiin ensin etsimään ratkaisua heille tarjotusta ohjeistusmateriaalista ja vasta tämän jälkeen pyytämään apua projektityöryhmäläisiltä. Ongelman ratkaisun aikana suunnittelijoilta oli tarkoitus tiedustella, miten he sovelsivat ohjeistusta ongelman ratkaisuun ja miksi he eivät pystyneet ratkaisemaan ongelmaa ohjeistuksen perusteella. Valitettavasti tapahtuman aikarajoitusten takia varsinainen työohje jouduttiin tekemään varsin yksityiskohtaiseksi, jotta osallistujat olisivat pystyneet sen avulla suoriutumaan suunnittelutehtävässä annetun ajan puitteissa. Tästä syystä sellaisia automaatioprofiilin ohjeistukseen liittyviä ongelmatilanteita ei ilmennyt, joissa edellä mainittuja ongelmatilanteisiin liittyviä kysymyksiä olisi voinut esittää. Ongelmatilanteiden kuvauksista saatiin kuitenkin osviittaa potentiaalisista kehitysprosessiin ja AP-työkalun käyttöön liittyvistä seikoista, jotka vaativat ohjeistusta.

Työnkulun aikana suunnittelijoita pyydettiin myös tutustumaan muutamiin kirjallisen ohjeistuksen kohtiin, joiden mallinnuselementtejä he sovelsivat työnkulussa. Näkemyksiä ohjeistuksen näistä kohdista kerättiin haastattelukysymyksillä, joita esitettiin osallistujille suunnitteluvaiheiden välissä. Tällä menetelmällä saatiin hieman enemmän arviointimateriaalia suunnittelijoilta, vaikka osa heistä ei ollutkaan tutustunut ohjeistuksen kuvauksiin käytettävistä mallinnuselementeistä, vaan soveltanut elementtejä intuitionsa varassa kokeilemalla. Positiivisena tässä voidaan nähdä tietenkin se, että suunnittelijat pystyivät intuitiivisesti päättelemään mallinnuselementtien käyttötarkoituksen tutustumatta niiden ohjeistukseen tai kuvauksiin. Huolimatta toivottua vähäisemmästä ohjeistukseen tutustumisesta, saatiin ohjeistusta koskevia arvioita muutamilta arviointitapahtumaan osallistuneilta suunnittelijoilta.

Ne suunnittelijat, jotka olivat lukeneet kirjallista ohjeistusta, pitivät ohjeistusta varsin ymmärrettävänä, eräs heistä jopa siinä määrin, että ohjeistus hänen mielestään olisi itsenäisenä riittävä kuvaus automaatioprofiilin käsitteiden hyödyntämiseen. Ohjeistukselle esitettiin kuitenkin myös kritiikkiä. Eräs suunnittelijoista mainitsi, että ohjeistus ohjaa käyttäjää vaikeaan suuntaan, asioita on esitetty hankalasti ja että ohjeistusta voisi yksinkertaistaa. Lisäksi hän esitti, että opiskelijoiden olisi hankala ymmärtää ohjeistusta. Valitettavasti haastattelumateriaalista ei selviä, olisiko suunnittelijalla ollut muita parannusehdotuksia ohjeistukseen yksinkertaistamisen lisäksi. Ohjeistuksessa paikoin esiintyvä hankala asioiden esittäminen voi kieliä heikosta kielellisestä ilmaisusta, asioiden huonosta käsittelyjärjestyksestä tai epäolennaisten asioiden esilletuomisesta ohjeistuksessa. Näitä seikkoja tulisi edellä esitettyjen arvioiden perusteella kuitenkin kehittää. Eräs mahdollisuus olisi pyytää suunnittelijoilta tutustumaan ohjeistukseen tarkemmin ja näin saada kattavampaa tietoa ohjeistuksen kehityskohteiden yksityiskohdista. Tämä voitaisiin toteuttaa esimerkiksi arviointitapahtuman tyyppisenä tilaisuutena, jossa oh-

jeistus tukeutuisi enemmän luvussa kuusi esitettyihin ohjeistusmuotoihin, eikä niinkään arviointitapahtumaa varten räätälöityyn työohjeeseen.

Toinen kokonaisuus, joka suunnittelijoiden kirjallista ohjeistusta koskevista arvioista nousi esille, oli AP-työkalun käyttöön liittyvät seikat. Eräs suunnittelijoista mainitsi, että kirjallisen ohjeistuksen kanssa ei pärjää, ellei omaa riittävää tietämystä AP-työkalun käytöstä. Tämä on totta, sillä kirjallisessa ohjeistuksessa ei käsitellä AP-työkalun käyttöä. Pitää kuitenkin muistaa, että kirjallisen ohjeistuksen tarkoituskaan ei ole neuvoa työkalun käytössä, vaan se keskittyy automaatioprofiilin käsitteiden ja niiden soveltamisen kuvaamiseen. Eräs arvioitsijoista ehdotti, että työkalun käyttöä voidaan ohjeistaa cheat sheet:ien avulla, eikä näitä asioita tarvitsisi välttämättä käsitellä laajasti kirjallisessa ohjeessa, kunhan ohje kertoo, miten työkalun käytössä pääsee alkuun. Joka tapauksessa automaatiosuunnittelijoiden arviointien perusteella ohjeistuksessa on puutteita AP-työkalun käytön osalta. Ohjeistuksen pitäisi näin ollen ottaa kantaa myös mallinnuselementtien käyttöön AP-työkalussa. Tämä on seikka, johon ohjeistus ei nykyisellään todellakaan ota mitenkään kantaa, mutta ohjeistuksen jatkokehityksessä tämä seikka tulee huomioida.

Haastatteluissa tiedusteltiin myös cheat sheet:ien ja wizardien käytöstä ja hyödyntämisestä yritysten omissa suunnitteluprojekteissa ja työkaluissa. Muutama suunnittelija esitti, että erityisesti wizardeille olisi mahdollisia käyttökohteita heidän nykyisin käyttämissään työkaluista ja kehitysprosesseissa. Eräs suunnittelijoista mainitsi, että wizardit olivat toimivia ja helpottivat monimutkaisten ja työläiden ongelmien ratkaisua, mihin wizardit on alun perin tarkoitettukin. Tältä pohjalta wizardeille voisi siis olla tilausta myös yritysten omien työkalujen ja kehitysprosessien puolella. Jos sopivia sovelluskohteita tulee esille, voidaan niiden toimintaa demonstroida AP-työkaluun tehtävillä toteutuksilla.

7.4. Jatkokehitysideoita

Kuten luvussa kuusi kävi ilmi, on ohjeistus vielä paikoin keskeneräinen. Tästä syystä pääasiallinen ohjeistuksen jatkokehitys liittyy puutteellisten kohtien täydentämiseen. Kirjallisessa ohjeistuksessa pitää panostaa erityisesti kehitysprosessin ohjeistukseen, jota ohjeistuksessa ei käsitellä vielä riittävän yksityiskohtaisella tasolla. AP-työkaluun integroidun ohjeistuksen osalta jatkokehitettävää on myös paljon. Cheat sheetien osalta on tehty muutama toimiva implementaatio, jotka ovat kuitenkin enemmänkin demonstraation omaisia kuin oikeasti suunnittelijan hyödynnettävissä olevia opasteita. Automaatiosuunnittelijoilta saadun palautteen perusteella cheat sheet:eille olisi kysyntää, joten niitä tulisikin toteuttaa lisää AP-työkaluun. Tarkoitus on myös toteuttaa muita kohdassa 6.4.4 esiteltyjä integroituja avustemuotoja AP-työkaluun.

Tällä hetkellä varsinkin kirjallinen ohjeistus nojaa varsin voimakkaasti automaatioprofiiliin, sen rakenteeseen ja elementteihin. Ohjeistuksessa on käyty läpi profiilin elementit samoin kuin profiilissa itsessäänkin. Profiilin käsitteistöstä kuitenkin suuri osa on abstrakteja käsitteitä, joita varsinaisessa mallinnustyössä ei käytetä lainkaan. Ohjeistus

on silti toteutettu UML:n lähtökohdasta niin, että käyttäjän tulee tuntea abstraktit käsitteet, ennen kuin hän voi täysin ymmärtää varsinaiset mallinnuskäsitteet. Tämä ei ole välttämättä automaatiosuunnittelijan kannalta paras mahdollinen rakenne. UML ei varsinaisesti korostu AUKOTON-kehitysprosessissa, joten ohjeistuksen ei välttämättä tarvitse perustua UML-lähtöisesti automaatioprofiiliin. Ohjeistuksessa automaatioprofiilin käsitteet voitaisiinkin esittää suunnittelijalähtöisesti siten, että kunkin mallinnuskäsitteen ohjeistus olisi kerättynä yhteen paikkaan. Tämä vähentäisi ohjeistuksen lukijalta vaaditun UML ymmärryksen määrää ja mahdollisesti suoraviivaistaisi ohjeistusta. Toisaalta on huomattava, että UML:n ja olioajattelun merkitys ja käyttö muiden, kuin automaatiotoimintojen mallintamisessa, puoltavat UML:n ja olioajattelun esilletuomista myös ohjeistuksessa.

Tärkeä seikka ohjeistuksen jatkokehityksen kannalta on kehitysprosessin, automaatioprofiilin ja AP-työkalun päivitykset. On selvää, että kaikki näistä tulevat päivittymään ja näin ollen muuttumaan nykyisestä muodostaan. Jo tätä kirjoitettaessa automaatioprofiiliin ollaan tekemässä päivityksiä Requirements- ja AutomationConcepts-aliprofiileihin, millä on vaikutuksensa myös AP-työkaluun. Nämä päivitykset on huomioitu diplomityössä, vaikka niitä ei automaatioprofiilin viralliseen versioon vielä olekaan lisätty. Koska ohjeistusta joudutaan päivittämään, voisi eräs sen jatkokehitysmahdollisuus olla ohjeistuksen siirtäminen ohjeistuskehitystyökaluun, ja lopettaa ohjeistuksen kehittäminen asiakirjamuodossa kokonaan. Tällöin päivitysten tekeminen olisi helpompaa, ja esimerkiksi AP-työkaluun integroitava HTML-pohjainen ohjeistus pystyttäisiin pitämään paremmin ajan tasalla.

Ohjeistuksesta projektityöryhmän jäsenille tehdyn kyselyn ja automaatiosuunnittelijoiden arvioiden pohjalta nousi esille lukuisia kirjallisen ohjeistuksen kehityskohteita. Ohjeistusta tulisi ensinnäkin yhtenäistää ja kehittää kieliasun osalta. Ohjeistuksesta toivottiin myös lyhennettyä versiota. Kirjallinen ohjeistus on nykyisellään noin 200 sivua pitkä, joten se on raskas. Ehdotettu lyhennelmä, joka sisältäisi esimerkiksi kehitysprosessin yleisen kuvauksen ja automaatioprofiilin esittelyn pakkaustasolla nähdään tekijän näkökulmasta erittäin hyvänä lisänä ohjeistuskavalkadiin. Lyhennetystä versiosta käyttäjä pystyisi nopeasti tarkistamaan unohtamiaan yksityiskohtia, eli se toimisi niin sanottuna tukiohjeena käyttäjälle, jolla olisi jo perustiedot kehitysprosessista ja sen käsitteistöstä. Lyhennetty versio olisi luonnollisesti myös yksinkertaisempi kuin ohjeistuksen nykyinen versio. Näin ollen se toteuttaisi erään arvioijan näkemyksen yksinkertaistetun ohjeistuksen paremmasta toimivuudesta. Lyhennetty versio ohjeistuksesta on erittäin toteutuskelpoinen, kunhan varsinainen ohjeistus on ensin saatu valmiiksi.

Arviointitilaisuudessa nousi myös esiin muutamia edellisissä kappaleissa mainitsemattomia ohjeistukseen liittyviä kehityskohteita. Eräs arvioijista sanoi, että paras tapa tarjota ohjeistusta työkalussa olisi mahdollistaa työkalun näkymässä esitettävistä elementeistä polku suoraan integroituun ohjeistukseen. Tässä käyttäjää avustavassa toiminnossa help-osion tietty sivu, esimerkiksi tietty automaatioprofiilin mallinnuselementtiä käsittelevä ohjesivu, linkitettäisiin työkalun editoriin ja siinä käsiteltäviin mallinnuselementteihin. Käyttäjä pääsisi suoraan editorin kautta tutustumaan haluamansa

mallinnuselementin ohjeistukseen help-osiossa. Tämä olisi varsin tehokas tapa tarjota käyttäjälle ohjeistusta, koska tällöin käyttäjän ei tarvitse erikseen navigoida ohjeistukseen, vaan hän pääsee siihen suoraan käsiksi. Koska tämän tyyppiselle avustemekanismille on tilausta, se voisi olla yksi työkalun ja ohjeistuksen jatkokehitystehtävistä.

Ohjeistuksesta saadun palautteen ja esille nousseiden jatkokehitysideoiden lukumäärän perusteella voidaan sanoa, että ohjeistuksessa riittää työsarkaa vielä pitkäksi aikaa. Koska AUKOTON-projekti on jo loppusuoralla, ei ohjeistukseen pystytä projektin puitteissa toteuttamaan kaikkia jatkokehitysideoita, joita edellä esitettiin. Näin ollen ohjeistuksen mahdollinen jatkokehitys tulee olemaan pitkälti AUKOTON-projektin perustalle rakentavien jatkohankkeiden varassa.

8. YHTEENVETO

AUKOTON-projekti kehittää uudenlaista lähestymistapaa automaation sovellussuunnitteluun. Automaatiosuunnittelu on perinteisesti ollut laitelähtöistä, mikä on hankaloittanut suunnittelutyön uudelleenkäyttöä erilaisilla alustoilla ja eri projekteissa. Ohjelmistotekniikan menetelmiä voidaan hyödyntää myös automaatiosuunnittelussa ja erityisesti automaation ohjelmistokehityksessä. Uusia tekniikoita ja menetelmiä tuleekin soveltaa automaatiosuunnitteluun, jotta se pysyy muuttuvan maailman mukana. Syitä tähän ovat esimerkiksi uusilla tekniikoilla ja menetelmillä saavutettava tehokkuus, laatu, kustannusten säästö pitkällä aikavälillä sekä toimintavarmuus ja luotettavuus.

AUKOTON-projektissa kehitettävä mallipohjainen automaatiosovellusten kehitysprosessi pyrkii vastaamaan automaatiosuunnittelun kehityskohteiden haasteisiin mallipohjaisella lähestymistavalla ja UML:ään pohjautuvalla mallinnuskäsitteistöllä. AUKOTON-kehitysprosessista voidaan erottaa kolme päävaihetta, joiden aikana mallinnetaan sovelluksen vaatimukset sekä toiminnot alustariippumattomasti ja -riippuvasti. Muunnokset mallien välillä on pääosin automatisoitu kuten myös sovelluskoodin generointi kohdealustalle alustariippuvan mallin perusteella. Näin pystytään saavuttamaan osa uusilla suunnittelumenetelmillä mahdollistuvista eduista.

Kehitysprosessissa sovelletut menetelmät ja tekniikat eivät ole automaatioalalla toistaiseksi yleisessä käytössä. Tästä syystä kehitysprosessille ja siinä hyödynnettävälle mallinnuskäsitteistölle toteutettiin tämän diplomityön puitteissa ohjeistus, jonka tavoitteena on maadoittaa uuden lähestymistavan käytäntöjä nykyisiin suunnittelukäytäntöihin. Näin halutaan mahdollistaa automaatiosuunnittelijoille pehmeämpi siirtyminen perinteisestä automaation suunnitteluprosessista AUKOTON-lähestymistavan mukaiseen mallipohjaiseen kehitysprosessiin.

Ohjeistusta toteutettiin sekä perinteiseen kirjalliseen, että AUKOTON-kehitysprosessissa hyödynnettävään AP-työkaluun integroituun muotoon. Kirjallisessa ohjeistuksessa käsiteltiin erityisesti AUKOTON-kehitysprosessin mallinnuskäsitteistöä, joka on määritelty UML-automaatioprofiilissa. Kirjallinen ohjeistus avaa automaatioprofiilissa määriteltyjä käsitteitä tarjoten esimerkkejä ja käytäntöjä niiden hyödyntämiseen AUKOTON-kehitysprosessissa. Esimerkeillä luodaan käyttäjälle näkemys mallinnuselementtien käytöstä ja ulkoasusta AP-työkalussa. Samalla ne havainnollistavat elementtien tyypillisiä käyttötapoja ja -kohteita. Kirjalliseen ohjeistukseen lisättävä laajempi kehitysprosessia kuvaava esimerkki tulee olemaan tärkeä osa ohjeistusta tarjoten kokonaiskuvan kehitysprosessin etenemisestä suunnittelijan kannalta.

AP-työkaluun integroitu ohjeistus keskittyy pääasiassa käyttäjän avustamiseen kehitysprosessin kannalta. AP-työkalun perustana oleva Eclipse-alusta tarjoaa monipuoliset

mahdollisuudet käyttäjän ohjeistukseen ja avustamiseen. Eclipsen rakenteen ansiosta lähinnä mielikuvitus asettaa rajoituksia tarjottavalle ohjeistukselle, sillä Eclipseä voidaan laajentaa liitännäisten avulla lähes rajoituksetta. AP-työkaluun integroitu ohjeistus toteutettiin kuitenkin Eclipsen tarjoamien valmiiden ohjeistus- ja avustemekanismien avulla, joista ohjeistuksessa sovellettiin Eclipsen help-osiota, cheet sheet:ejä ja wizardeja.

Eclipsen help-osiota laajennettiin kirjallisen ohjeistuksen materiaalilla, jolla help laajennettiin käsittelemään myös automaatioprofiilin käsitteistöä. Näin käyttäjille tarjottiin mahdollisuus saada nopeasti ohjeistusta automaatioprofiilin käsitteisiin liittyen. Cheat sheet:eilla kuvattiin AUKOTON-kehitysprosessin vaiheet korkealla abstraktiotasolla. Tällä pyrittiin tukemaan käyttäjien itsenäistä työskentelyä AUKOTON-kehitysprosessissa tarjoamalla yksinkertainen osatehtäviin perustuva listaus kehitysprosessin vaiheista. Wizardilla toteutettiin säätöpiirin luomisen avustetoiminto, jonka avulla voidaan generoida säätöpiirin runko toiminnalliseen malliin. Wizardin avulla mahdollistettiin suunnitteluajan tehokkaampi käyttö vähentämällä käyttäjän suorittamaa manuaalista työtä.

Ohjeistuksesta saatiin automaatiosuunnittelijoilta ja AUKOTON-projektityöryhmän jäseniltä pääasiassa positiivista palautetta. Tämän perusteella voidaan katsoa, että ohjeistuksen perusrakenne on kunnossa. Arvioinneissa nousi esille myös hyviä jatkokehitys- ja parannuskohteita ohjeistukseen liittyen. Tietäen, että ohjeistukseen ollaan oltu pääosin tyytyväisiä, on esitettyjen parannus- ja kehitysehdotusten perusteella hyvä jatkaa ohjeistuksen kehittämistä entistä paremmin automaatiosuunnittelijoita ja muita käyttäjiä palvelevaksi kokonaisuudeksi.

LÄHDELUETTELO

[1] Dorf, R. & Bishop R. Modern Control Systems. 10th edition. Upper Saddle River (NJ) 2005, Pearson Prentice Hall. 881 p.

[2] Halmetoja, J. Rakenteisten dokumenttimuotojen käyttö automaatiosuunnittelussa. Diplomityö. Tampere 2000. Tampereen teknillinen korkeakoulu, Automaatio- ja säätötekniikan laitos. 76 s.

[3] Judén, T. Automaation suunnitteluprosessin tukeminen integroidulla taitotukisovelluksella. Diplomityö. Tampere 2007. Tampereen teknillinen yliopisto, Automaatiotekniikan koulutusohjelma. 52 s.

[4] Asmala, H., Koskinen, K., Koskela, M., Mätäsniemi, T., Soini, A., Strömman, M., Tommila, T. & Valkonen, J. Automaatiosovellusten ohjelmistokehitys - Suunnittelun työtavat, välineet ja sovellusarkkitehtuurit. Helsinki 2005, Suomen automaatioseura ry. 152 s.

[5] Viinikkala, M. Integration of automation design information using XML technologies. Master of Science Thesis. Tampere 2002, Tampere University of Technology, Institute of Automation and Control. 115 p.

[6] Hästbacka, D., Mätäsniemi, T., Unifying Process Design with Automation and Control Application Development - an Approach Based on Information Integration and Model-driven Methods. 13th IFAC Symposium on Information Control Problems in Manufacturing, Moscow, Russia, June 3-5, 2009. Accepted paper.

[7] HSE. Out of control - Why control systems go wrong and how to prevent failure. Sudbury 1995, Health & Safety Executive (HSE)/HSE Books. 65 p.

[8] Ajo, R., Hakonen, S., Harju, H., Järvi, J., Kaskes, K., Lenardic, E., Niukkanen, E., Nurminen, T., Ritala, P., Tolppanen, M. & Tommila, T. Laatu Automaatiossa - Parhaat käytännöt. Helsinki 2001, Suomen Automaatioseura ry. 245 s.

[9] Rauhamäki, J., Hästbacka, D., Vepsäläinen, T., Kuikka, S., Tommila, T., Mätäsniemi, T., Peltola, J. & Sierla, S. UML-mallipohjainen kehitysprosessi automaatio-ohjelmistoille. Automaatio XVIII Seminaari, Helsinki 17-18.3.2009. Helsinki 2009, Suomen Automaatioseura ry.

[10] Kent, S. Model Driven Engineering. Proceedings of the IFM conference on integrated formal methods, Turku, Finland May 2002. Berlin/Heidelberg 2002, Springer. pp. 286-298.

[11] Stahl, T., Völter, M., Bettin, J., Haase, A. & Helsen, S. Model-Driven Software Development. 1st edition. Heidelberg 2005 John Wiley & Sons, Ltd, 444 p.

[12] Brown, A. Model driven architecture: Principles and practice. Software and Systems Modeling 3(2004)4, pp. 314-327.

[13] Partanen, T. MDA:n hyödyntäminen ohjelmistokehityksessä. Diplomityö. Tampere 2005. Tampereen teknillinen yliopisto, Tietotekniikan osasto. 59 s.

[14] Vepsäläinen, T. UML-profiilityökalu automaatiosuunnitteluun. Diplomityö. Tampere 2008. Tampereen teknillinen yliopisto, Automaatiotekniikan koulutusohjelma, 167 s.

[15] Swithinbank, P., Chessell, M., Gardner, T., Griffin, C., Man, J., Wylie, H. & Yusuf, L. Patterns: Model-Driven Development Using IBM Rational Software Architect [WWW]. International Technical Support Organization. 2005, [viitattu 10.2.2009]. 252 p. Saatavissa: http://www.redbooks.ibm.com/redbooks/pdfs/sg247105.pdf.

[16] Object Management Group. MDA Guide Version 1.0.1 [WWW]. 12.6.2003, [viitattu 16.1.2009]. Saatavissa: http://www.omg.org/docs/omg/03-06-01.pdf.

[17] Lang, U. & Schreiner, R. Developing Secure Distributed Systems with CORBA. Norwood 2002, Artech House Publishers. 308 p.

[18] Kleppe, A., Warmer, J., & Bast, J. MDA Explained: Practice and Promise. Boston 2003, Addison-Wesley Professional.192 p.

[19] Tommila, T. & Mätäsniemi, T. AUKOTON data model. 7.4.2009, VTT. Unpublished document. 49 p.

[20] Purdum, J. Visual Basic® .Net Primer plus. 2003, Sams Publishing. 672 p.

[21] Fowler, M. UML distilled: a brief guide to the standard object modeling language. 3rd edition. 2003, Addison-Wesley Professional. 208 p.

[22] Object Management Group. Introduction To OMG's Unified Modeling Language [WWW]. 18.6.2009 [viitattu 24.8.2009]. Saatavissa: http://www.omg.org/gettingstarted/what_is_uml.htm#12DiagramTypes

[23] Pilone, D & Pitman, N. UML2.0 in a nutshell. 2nd edition. Sebastopol 2005, O'Reilly Media, Inc. 234 p.

[24] Douglass, B. Real time UML: advances in the UML for real-time systems. 3rd edition. 2004 Boston, Pearson Education, Inc. 752 p.

[25] Kuikka, S. Luentokalvot: UML RT profiili [WWW]. Systeemitekniikan laitos. 22.10.2008, [viitattu 24.3.2009]. Saatavissa: http://www.ac.tut.fi/aci/courses/ACI-32020/RAJOKalvot8S2008.pdf.

[26] OMG Adopted Specification formal/05-01-02. UML Profile for Schedulability, Performance and Time Specification. 2005, Object Management Group. 235 p.

[27] OMG Adopted Specification formal/2008-11-01. OMG Systems Modeling Language (OMG SysML). 2008, Object management Group. 256 p.

[28] Ritala, T. UML Automation Profile Specification (version 0.14). 12.2.2006, Tampere University of Technology. Unpublished UML profile. 82 p.

[29] OMG Adopted Specification formal/2008-04-05. UML Profile for Modeling Quality Of Service and Fault Tolerance Characteristics and Mechanisms. 2008, Object Management Group. 106 p.

[30] Rauhamäki, J. User guide for the UML Automation profile (version 0.4.0). Tampere 1.4.2009, Tampere University of Technology. Unpublished guidance. 159 p.

[31] Alho, P. Application of New Automation Software Design and Integration Technologies in Teaching. Master of Science Thesis. Tampere 2009. Tampere University of Technology, Department of Automation Science and Engineering. 95 p.

[32] Vepsäläinen, T., Hästbacka, D. & Kuikka, S. Tool Support for the UML Automation Profile - for Domain-Specific Software Development in Manufacturing. The Proceedings of the 3rd International Conference on Software Engineering Advances, Sliema, Malta, October 26-31, 2008. Los Alamitos, California 2008, IEEE Computer Society. pp. 43-50.

[33] Holzner, S. Eclipse. 1st edition. Sebastopol 2004, O'Reilly Media, Inc. 334 p.

[34] KW-Software. MULTIPROG®: Modern and Powerful IEC 61131 Programming [WWW]. [viitattu 15.4.2009]. Saatavissa: http://www.kw-software.com/com/index1024.html.

[35] Tommila, T., Peltola, J., Kuikka, S., Mätäsniemi, T., Hästbacka, D. & Vepsäläinen, T. Seamless development path for automation applications – Concepts and approach, an overview. 3.3.2009, AUKOTON-project group. Unpublished document. 36 p.

[36] Automaatiosovelluksen yhtenäinen kehityspolku – PI-kaaviosta ohjelmistototeutukseen (AUKOTON). 12.9.2007, AUKOTON-projektityöryhmä. Julkaisematon tutkimussuunnitelma. 20 s.

[37] IEC 61804-1. Standard data element types with associated classification scheme for electric components - Part 1: Definitions – Principles and methods. Geneva 2003, International Electrotechnical Commission. 134 p.

[38] IEC 62424. Representation of process control engineering - Requests in P&I diagrams and data exchange between P&ID tools and PCE-CAE tools. 2008, International Electrotechnical Commission. 135 p.

[39] Peltola, J., Tommila, T., Mätäsniemi, T., Vepsäläinen, T., Hästbacka, D. & Rauhamäki, J. Aukoton Demo Event 11.12.2008. 11.12.2008, AUKOTON project group. Unpublished presentation slides.

[40] Vepsäläinen, T., Hästbacka, D., Rauhamäki, J., Kuikka, S., Peltola, J., Sierla, S., Papakonstantinou, N., Mätäsniemi, T. & Tommila, T. Työkaluketju automaatiosovellusten UML-mallipohjaiseen suunnitteluun. Automaatio XVIII Seminaari, Helsinki 17-18.3.2009. Helsinki 2009, Suomen Automaatioseura ry.

[41] IEC 61131-3. Programmable controllers – Part 3: Programming languages. 2003, International Electrotechnical Commission.

[42] Metso Automation. Products - FbCAD - Function block diagram CAD [WWW]. 2008, [viitattu 3.2.2009]. Saatavissa: www.metso.com/Automation/ip_prod.nsf/WebWID/WTB-070122-2256F-16719?OpenDocument.

[43] Tiedt, P. Building cheat sheets in Eclipse. IBM developerWorks [WWW]. IBM. 13.12.2005, [viitattu 7.4.2009]. Saatavissa: https://www6.software.ibm.com/developerworks/education/os-cheatsheets/os-cheatsheets-a4.pdf.

[44] Adobe Systems Incorporated. Introducing Adobe® RoboHelp® 8 [WWW]. [viitattu 24.8.2009]. Saatavissa: http://www.adobe.com/products/robohelp/.

LIITE 1: KATKELMA KIRJALLISESTA OHJEISTUKSESTA

Tässä liitteessä on esitetty lyhyt katkelma kirjallisesta ohjeistusdokumentista ”User guide for the UML Automation Profile” [30]. Katkelman tarkoituksena on antaa diplomityön lukijalle kuva ohjeistuksen yleisestä ilmeestä ja kirjoitustyylistä. Tavoitteena on myös tuoda esiin se tapa, jolla ohjeistus pyrkii maadoittamaan automaatioprofiilin käsitteistön AP-työkaluun ja esittämään sen sovellusmahdollisuuksia. Liitteen katkelma on ohjeistusdokumentin luvusta kolme, joka kuvaa Requirements-aliprofiilin mallinnuselementtejä. Tekstissä esiintyvä viite (Ritala, 2006) vastaa tämän diplomityön viitettä [28]. Liitteessä esitettävä katkelma pohjautuu lähes kokonaisuudessaan automaatioprofiilin määrittelyyn [28], vaikka se ei varsinaisessa katkelmassa tulekaan esiin. Huomaa, että ohjeistuskatkelman asettelu on otettu ohjeistusdokumentista, joten se eroaa diplomityön varsinaisten lukujen asettelusta, vaikka asettelua onkin korjattu muistuttamaan tämän diplomityön ulkoasua. Viittaukset alilukuihin ovat valideja ainoastaan tämän liitteen sisällä.

3.1 RequirementModeling

The RequirementModeling package enables user to model general, functional and non-functional requirements and includes generic concepts for requirement specification. Figure 11 represents an introductory example of the RequirementModeling usage, which includes various concepts of the package. The concepts are used to model general requirements of the system. The user is able to group requirements, model requirements, detail them with additional elements (e.g. TraceRelation) and create relations between requirements. Other concepts are also included but these are introduced in the forthcoming chapters.

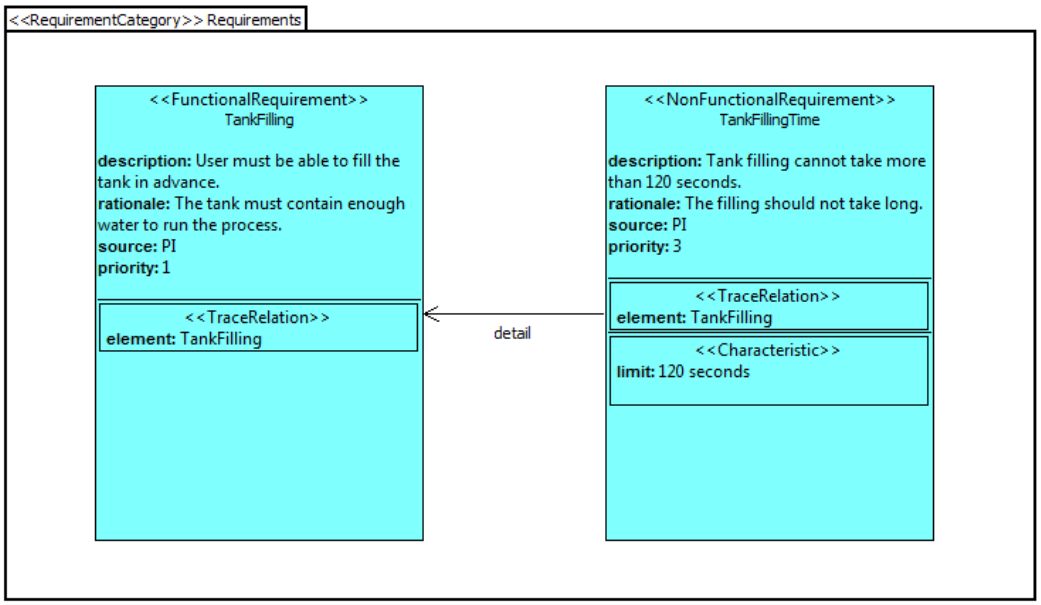


Figure 11: RequirementModeling example

3.1.1 RequirementCategory

A Requirement category is the groundwork for requirements. The Requirement category is used to gather requirements according to some criteria i.e. requirements that share a common issue can be collected under the same requirement category.

The requirements are easier to manage, if they are categorized in a sensible way. The UML Automation Profile offers the RequirementCategory element to categorize requirements. When requirements are categorized the designer must decide how the categorization is done and there are several ways to divide the criterions which define the categories. One possibility is to have general categories as functional requirements (what the system should do), non-functional requirements (e.g. restriction, prerequisites and conditions to functionality (Ritala, 2006)) and safety requirements. This kind of categorization is practical for small entireties such as the example process or similar system e.g. condenser with temperature control. An example of defined categorization is given in Figure 12.

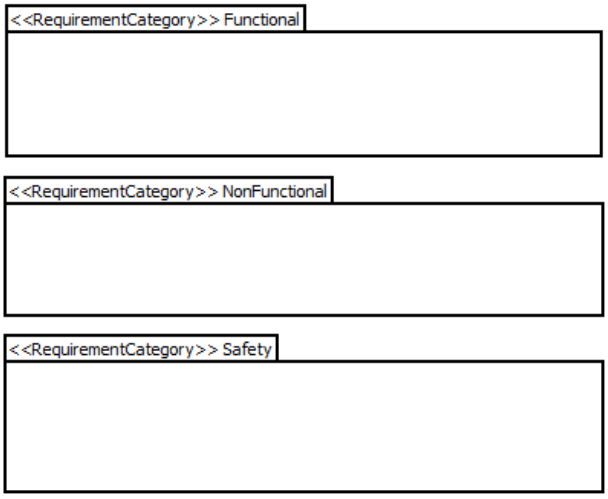


Figure 12: A requirement category division example

Notice that in Figure 12 the words “Functional”, “NonFunctional” and “Safety” are names of the elements, not the criterions. An example of the criterion in Figure 12 is given in Figure 13. The distinction between the name of the category and the criterion of the category is noticeable. The criterion-property is the actual property that describes the categorization criterion not the name of the property.

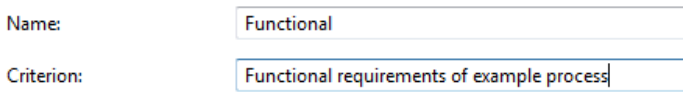


Figure 13: RequirementCategory properties in UML AP tool

Another way to divide the categories is by physical or logical division of the system. This kind of approach is suitable for most of the cases. The example process of this

guide is not quite large enough to be categorized this way, but consider for example paper machine requirement categories divided into a head box, machine clothing, a press section, a dryer section and calendar section. These categories may be further divided into more detail categories e.g. using functional, non-functional and safety related approach as mentioned above. In Figure 14 requirements for the head box are further divided into more detail categories. Requirements for other physical entities of the paper machine could also be modelled in the same way as requirements for the head box.

The categorization depends of the size of system. Small systems do not need many categories (possibly just one), while in large systems there might be dozens of categories.

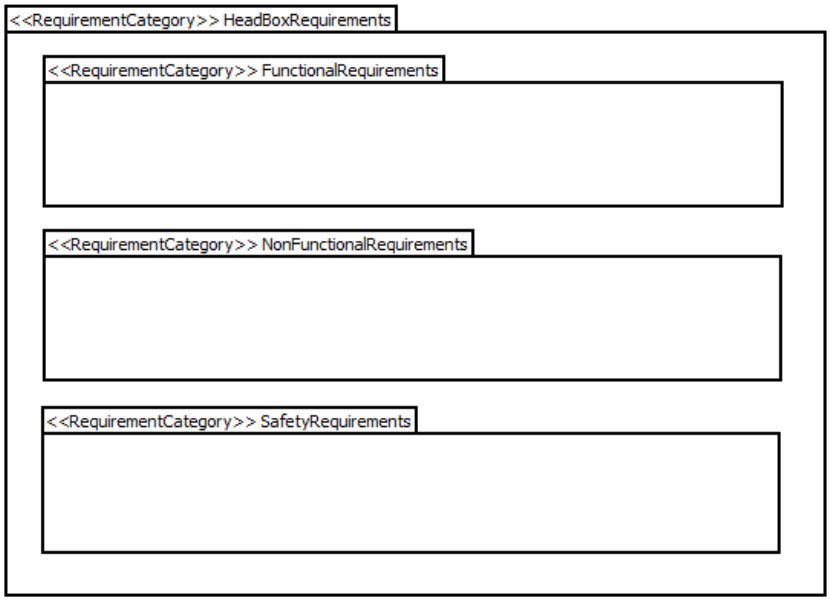


Figure 14: Nested requirement categories

Any kind of rational categorization of the requirements is valid. One must also remember that the diagram itself is categorizer and that also has an impact on categorization. This matter will be discussed in chapter 7.1.

Properties:	
criterion	A category can be specified one or more criteria that describe the point of view of the category.
Associations:	
Requirement	A requirement category organizes a set of requirements. *Multiple requirements and/or requirement categories can be inserted on a RequirementCategory element.*

Table 2: RequirementCategory specification

3.1.2 Requirement

“Requirement is a generic concept for capturing various kinds of requirements and hierarchies or requirements that must be satisfied by the system under design” (Ritala, 2006). Requirement is an essential element in the requirement sub-profile. Many other elements are inherited from the Requirement element, that is, the Requirement is a basis of other element as well as independent modelling object. In forthcoming chapters we will see that e.g. FunctionalRequirement and NonFunctionalRequirement are inherited from the Requirement element.

Requirement is textual description of property that the system should fulfil. Requirements are mostly used to define high-level requirements such as plant and process requirements. More definite requirements are recommended to be given in functional requirements and non-functional requirements, which are discussed later. Each requirement must lie on a RequirementCategory-element. An example is provided in Figure 15 in which the high-level requirement for the example process (see1.4) is defined. Since the requirement in Figure 15 is a high-level requirement, the definition and the rationale is taken straight from the process engineering step.

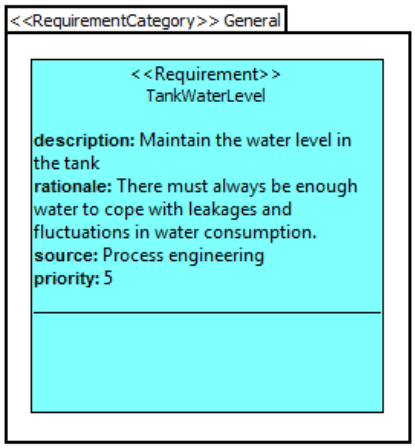


Figure 15: Requirement on RequirementCategory-element

Properties:	
Id	An identifier for the requirement. *Any kind of identifier is valid, but each requirement must have unique id.*
description	An informal description of the property captured by the requirement.
source	An optional list of the stakeholders driving this requirements Stakeholders could be for example customer, designer or same previous design phase such as Process engineering.
rationale	An informal description of the reasons within the background of the requirement, such as laws, regulations, knowhow, etc. *Other rationales could be for example a customer need or problem with current design* (Asmala, et al., 2005).
priority	The priority of the requirement, specifying the urgency of implementing the part of the software that satisfies the requirement. *Any kind of priority classification is valid e.g. priority from 1 to 5 where 5 is the most urgent and 1 is the least urgent requirement to be implemented.*
Associations:	
Requirement	A requirement may relate to other requirements in various manners (see RequirementRelation).
ModelElement	A requirement may be related with a modelling element that is used for satisfying the requirement for verification purposes (see TraceRelation).

Table 3: Requirement-element specification

3.1.3 FunctionalRequirement

Functional requirements define what the system should do i.e. captures behavioural and functional properties of the system. Functional requirements are used for high- and low-level requirements. Especially the functional requirements should be preferred over standard requirements while modelling low-level functional requirement.

As the Figure 10 states, the functional requirement is inherited from requirement, which indicates that functional requirement has all the properties that a requirement has. In the Automation Profile there is a minor (yet distinct) difference between a requirement and a functional requirement; a functional requirement can be supplemented with a functional description (see 3.1.4). In brief, a functional requirement is a requirement with an extra ability (functional description) and different stereotype and it is solely used to capture functional requirements.

Figure 16 shows some of the functional requirements of the example process. The main requirement is the TankLevelControl-requirement, which states the basic requirement of the controllable water level in the tank. The other two functional requirements detail the TankLevelControl-requirement. This matter will be discussed in chapter3.1.7.

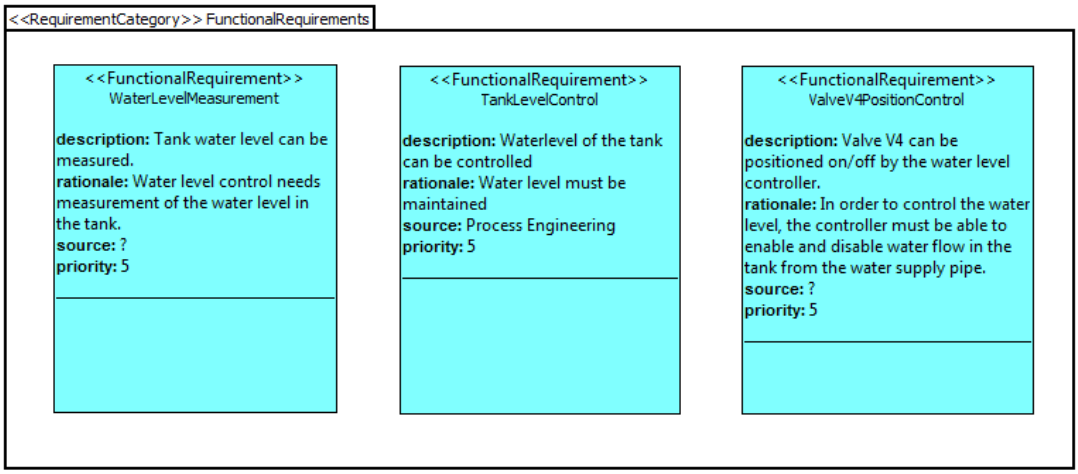


Figure 8.1: Some of the functional requirements of the example process

Properties:	
-	-
Associations:	
Functional Description	A suitable collection of descriptions that specify the functionality captured by the requirement.

Table 4: FunctionalRequirement specification. Notice, that FunctionalRequirement inherits all the properties of the Requirement.

3.1.4 FunctionalDescription

Functional descriptions are used in situations where textual presentation is insufficient for capturing the requirement. “In addition, various kinds of graphical descriptions, such as use cases or state machines, provide intuitive means for capturing the behaviour related with a functional requirement.” (Ritala, 2006). The UML itself contains many diagram types (including the two mentioned above), which can be used with the Automation Profile as a reference as described here.

FunctionalDescription is not (yet) implemented in the UML AP tool as an independent model element (like e.g. TraceRelation, which has its own modelling block). Nevertheless, the FunctionalDescription can be used through comment element (in fact all unimplemented modelling elements are used this way). An example is given in Figure 17. In the example the water tank from the example process is considered (although this is not defined functionality of the example process). The designer wants to describe a requirement about the states of the water tank. This is complicated task to present accurately by reasonable amount of textual notation. Instead a UML state machine diagram is used to describe the state behaviour of the tank. The state machine diagram “WaterTankStates” can be attached as a part of the requirement using the “<<FunctionalDescription>>” stereotype in a comment (the yellow field). The comment field must include the “<<FunctionalDescription>>” stereotype, the name of the element “WaterTankStateBehavior” and the name of the referred document “WaterTankStates”, in order to be a valid FunctionalDescription.

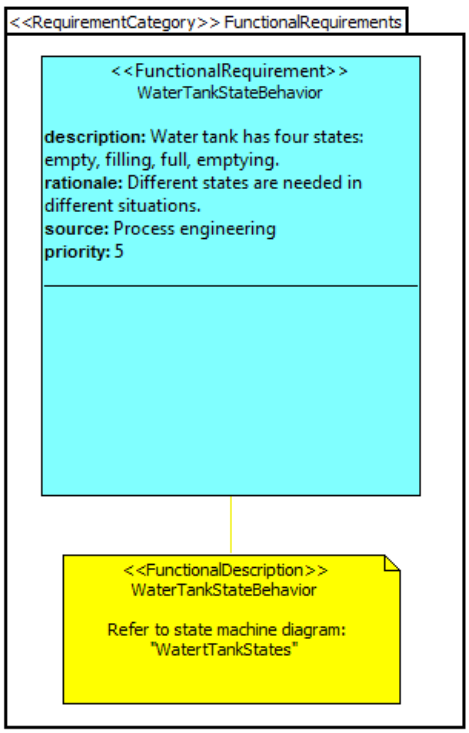


Figure 17: Example of FunctionalDescription

Properties:	
-	-
Associations:	
-	- *The reference to external document may be though as an association since it points out an element (document), but it is not an association in the Automation Profile scope.*

Table 5: FunctionalDescription specification

3.1.5 NonFunctionalRequirement

Non-functional requirements are constrains or qualitative requirements of the system behaviour i.e. additional constrains to the functional requirements e.g. time, performance, usability, etc. restrictions. In other words, non-functional requirements specify the functional requirements. Non-functional requirements are used for high- and low-level requirements. Especially the non-functional requirements should be preferred over standard requirements while modelling low-level non-functional requirements.

Non-functional requirements are inherited from the requirement in the same way as functional requirements, so non-functional requirements have the same properties as requirements. Again a minor difference exists; instead of a functional description, a characteristic-property can be supplemented with the non-functional requirement.

In Figure 18 two non-functional requirements of the example process are presented. The requirements are derived from the textual requirement presentation of the example process. It can be seen that the non-functional requirements in the example detail the

functional requirement presented in Figure 16. These requirements can be connected, since they are related (see RequirementRelation3.1.8).

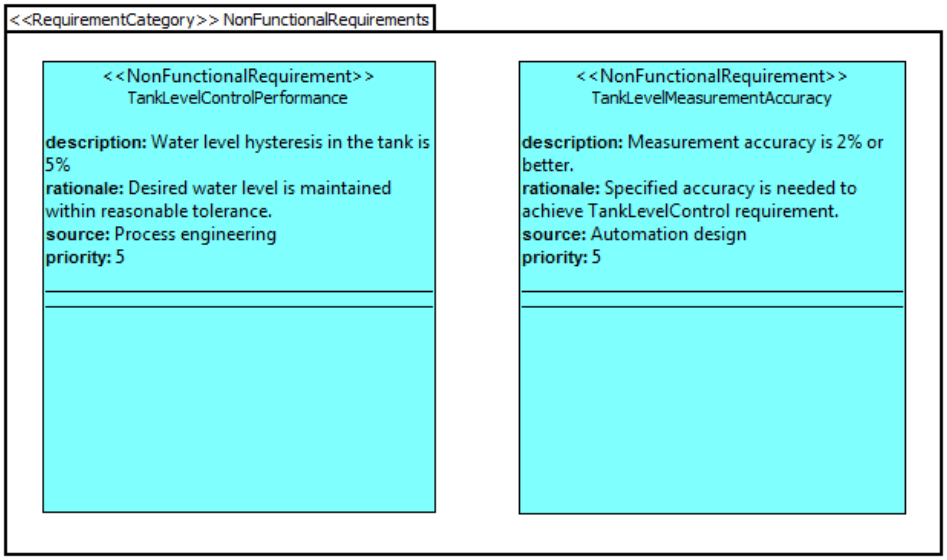


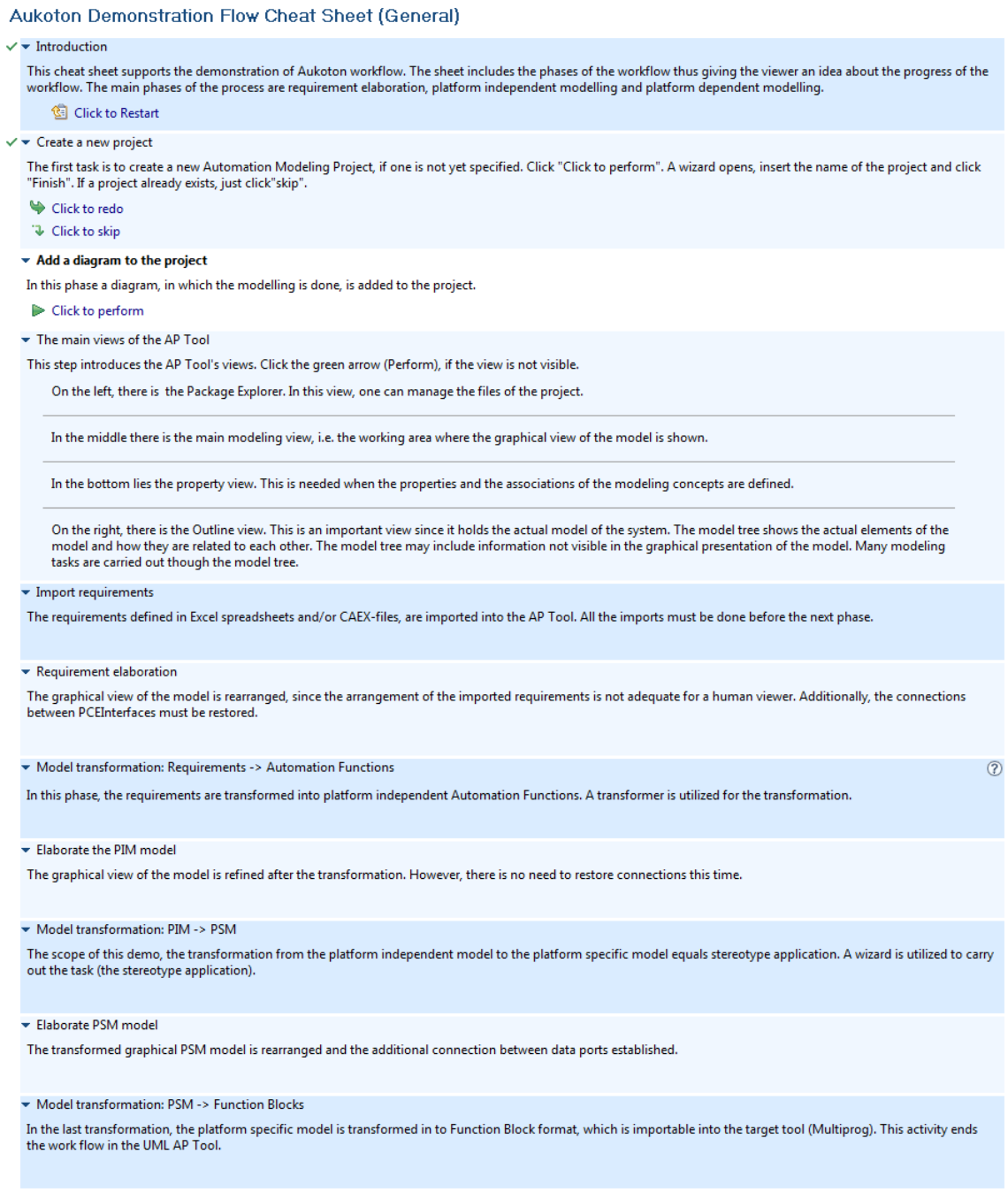
Figure 18: Some non-functional requirements of the example process

Properties:	
-	- Notice, that NonFunctionalRequirement inherits all the properties of the Requirement.
Associations:	
Characteristic	The non-functional characteristics captured by the requirement.

Table 6: NonFunctionalRequirement specification

LIITE 2: AUKOTON-KEHITYSPROSESSIA KUVAAVA CHEAT SHEET

Huomaa kohdassa ”Model transformation: Requirements → Automation Functions” oleva ympyröity kysymysmerkki, joka viittaa linkkiin Eclipsen Help-osioon. Kyseisen linkin takaa on siis saatavissa lisätietoja esimerkiksi transformaatiosta.

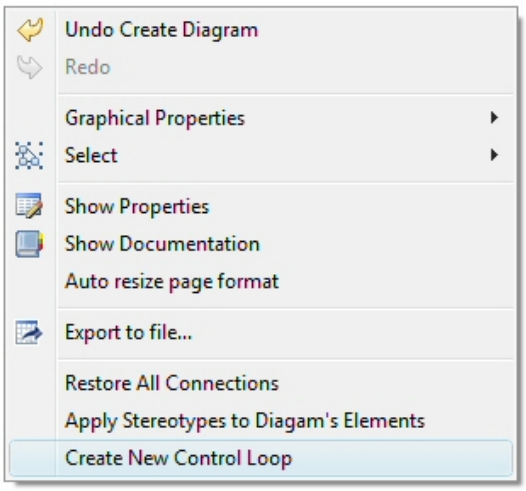


LIITE 3: CONTROL LOOP CREATOR WIZARDIN KÄYTTÖTAPAUS

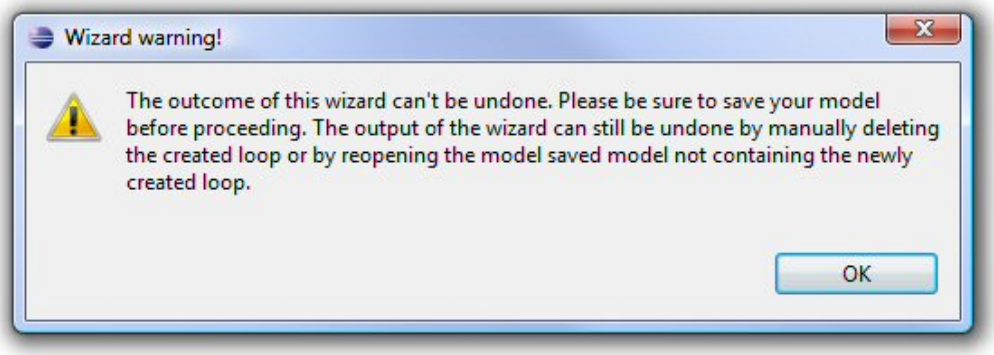
Tässä liitteessä kuvataan eräs Control loop creator wizardin käyttötapaus. Liitteessä sovelletaan automaatioprofiilin versiota 0.14 siihen liittyvine päivityksineen ja AP-työkalun versiota 0.9.2, joten esitetyt stereotyypit ja kaaviokuvat eroavat muualla diplomityössä esitetyistä vastineistaan.

Esimerkkitapauksena käsitellään tilanne, jossa käyttäjä haluaa luoda malliin uuden säätösilmukan (ControlLoop), koska sitä ei ole generoitunut transformaatiossa vaatimusmallista alustariippumattomaan malliin. Tarkoituksena on tuottaa alustariippumaton säätösilmukka, joka säätää tankin pinnan korkeutta. Säätösilmukkaan tarvitaan kaksi mittaussisääntuloa yksi säätöalgoritmi ja yksi ohjauslähtö. Lisäksi ohjauslähtö tarvitsee lukituksen, jolla estetään tankin pinnankorkeuden ajautuminen liian korkealle.

Wizard käynnistetään avaamalla sopiva säätörakennekaavio ja painamalla hiiren oikeaa näppäintä ja valitsemalla valikosta ’Create New Control Loop’. Kuvassa 1 yksi on esitetty wizardin avaaminen edellä selostetulla tavalla. Ennen wizardin käynnistymistä käyttäjää varoitetaan siitä, että wizardin generoimaa mallin osaa ei voi kumota, eli käyttäjä ei voi poistaa luotuja osia kumoamalla toimintoa. Kuvassa 2 on esitetty ponnahdusikkuna, jossa käyttäjää varoitetaan.

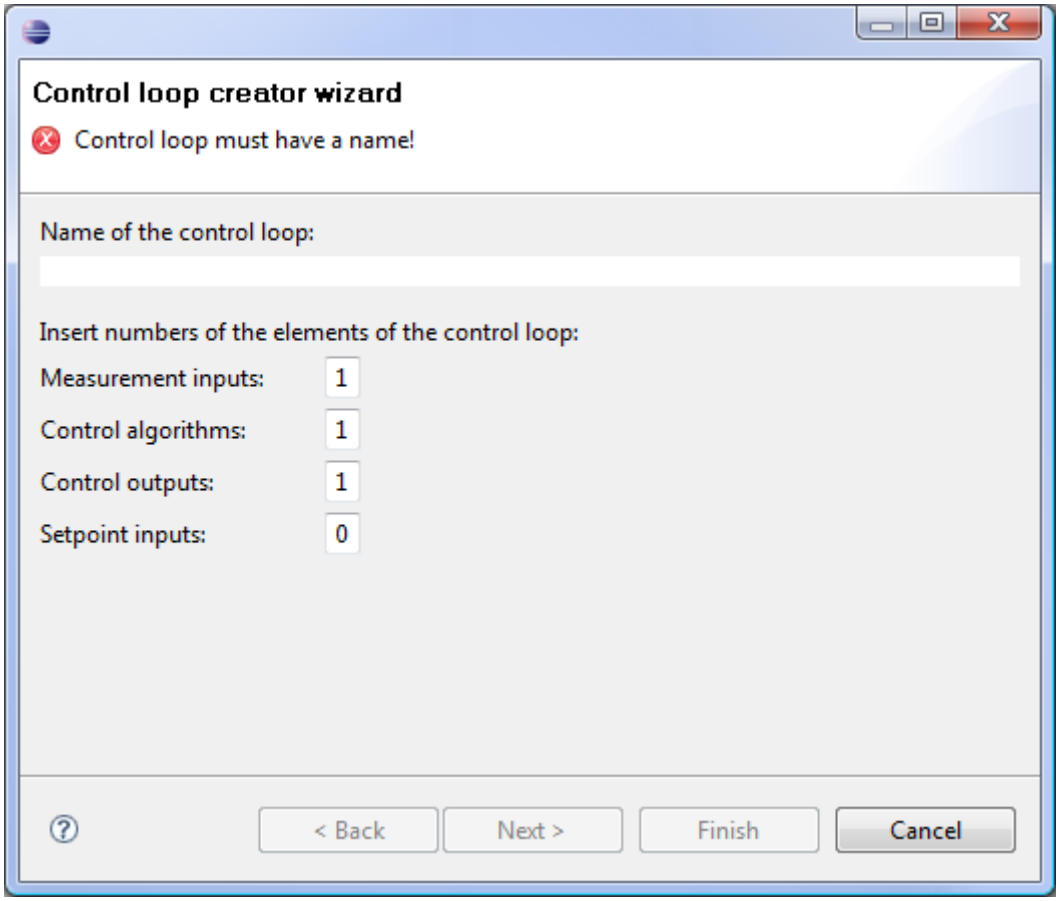


Kuva 1. *Control loop creator wizardin käynnistäminen*



Kuva 2. *Varoitus toiminnon kumoamattomuudesta ja kehotus tallentaa nykyinen malli*

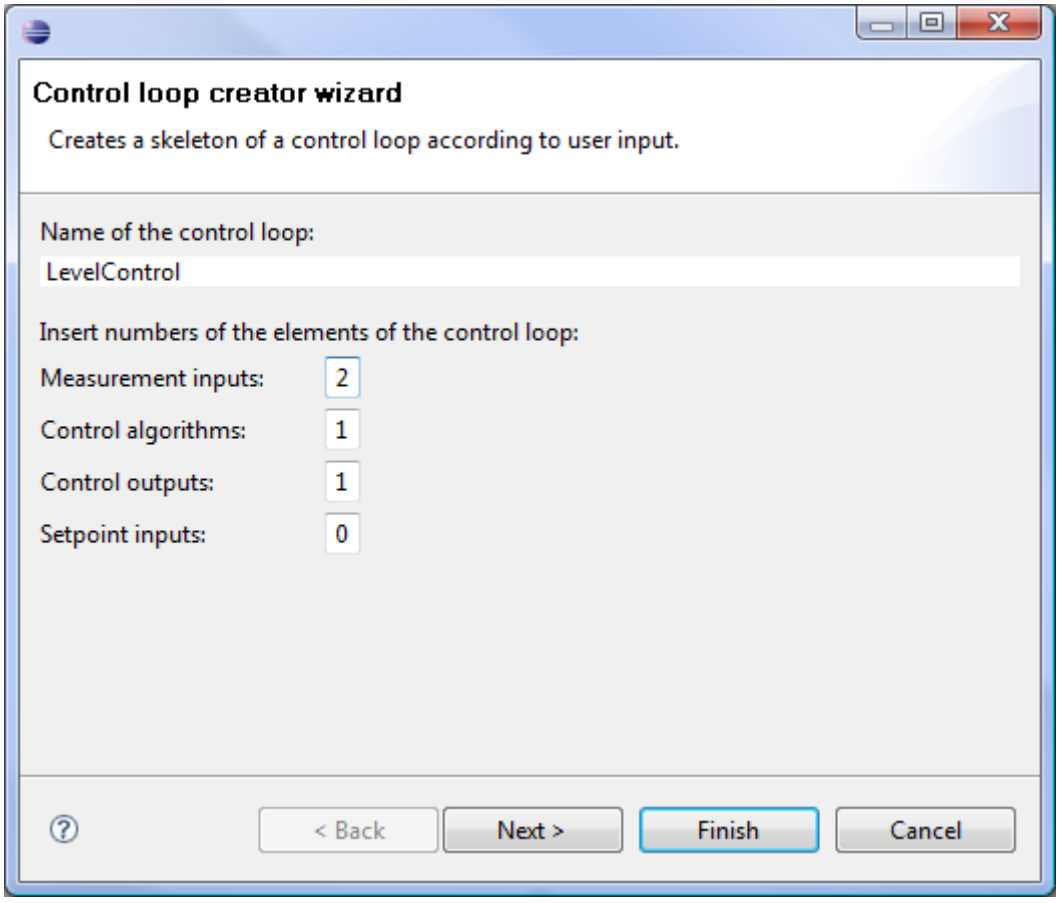
Käyttäjän tallennettua mallinsa ja kuitattuaan varoituksen, hän pääsee varsinaisen wizardin aloitussivulle. Aloitussivu on esitetty kuvassa 3. Kuvan 3 tilanteessa käyttäjä on poistanut wizardin ehdottaman nimen säätösilmukalle, mutta ei ole vielä ehtinyt syöttää uutta nimeä. Tästä aiheutuu lukitustila, jossa wizard vaatii käyttäjää syöttämään silmukalle nimen, ennen kuin se sallii käyttäjän jatkaa wizardin suorittamista.



Kuva 3. *Virheellisestä syötteestä aiheutunut wizardin lukkiutumistila*

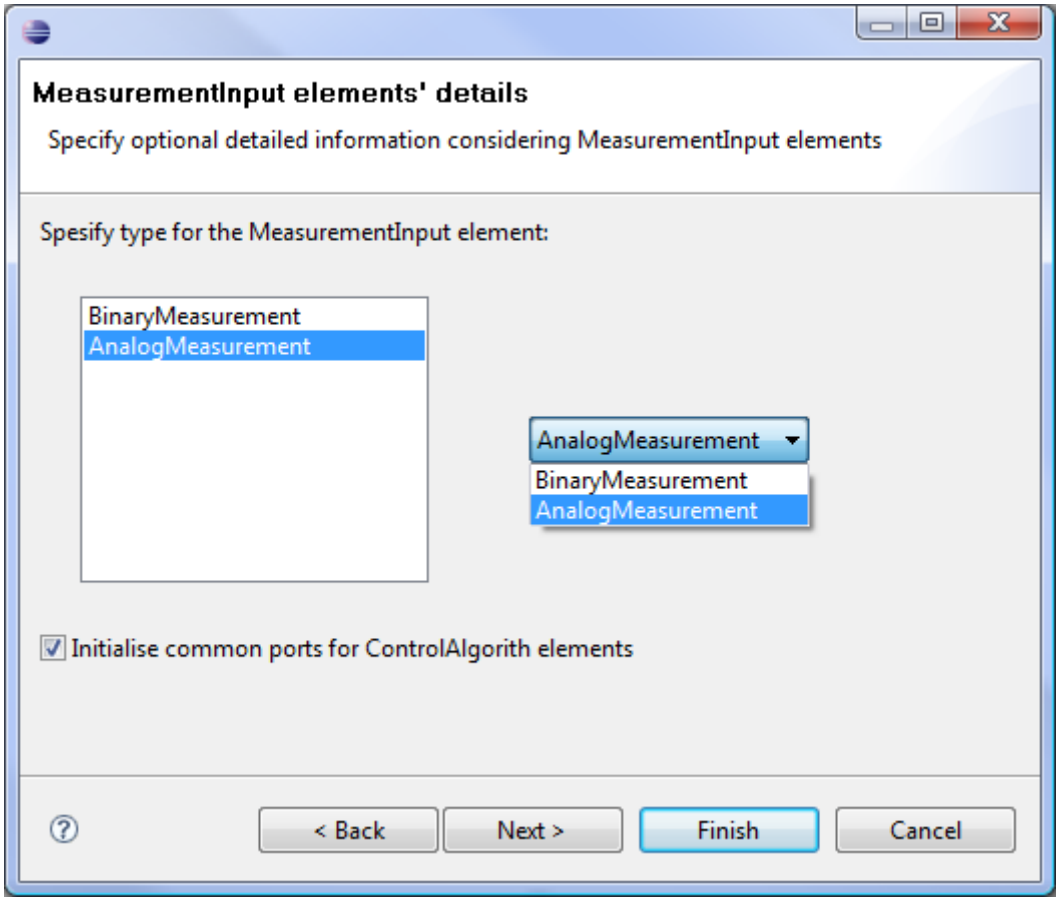
Kuvan 4 tilanteessa käyttäjä on antanut generoitavalle säätösilmukalle nimen, jolloin lukitus on purkautunut. Lisäksi käyttäjä on muuttanut mittaussisäänmenojen lukumääräksi kaksi, kuten oli tarkoitus. Asetusarvoporttien oletusarvoinen määrä on nolla. Näitä portteja käytetään, jos säätöpiiriä halutaan ajaa rajapinnan läpi esimerkiksi jotain toisesta säätösilmukasta niin, että korkeamman tason silmukka tarjoaa generoitavalle silmukalle asetusarvon.

Jos käyttäjä haluaisi generoida säätösilmukan wizardin oletusasetuksilla, hän voisi tehdä sen jo tässä näkymässä annettujen tietojen perusteella. Tässä esimerkkitapauksessa käyttäjä kuitenkin generoi lopputulokset vasta kun hän on antanut kaikille elementeille tarkentavat tiedot. Näin ollen käyttäjä siirtyy seuraavalle wizardin sivulle valitsemalla Next.



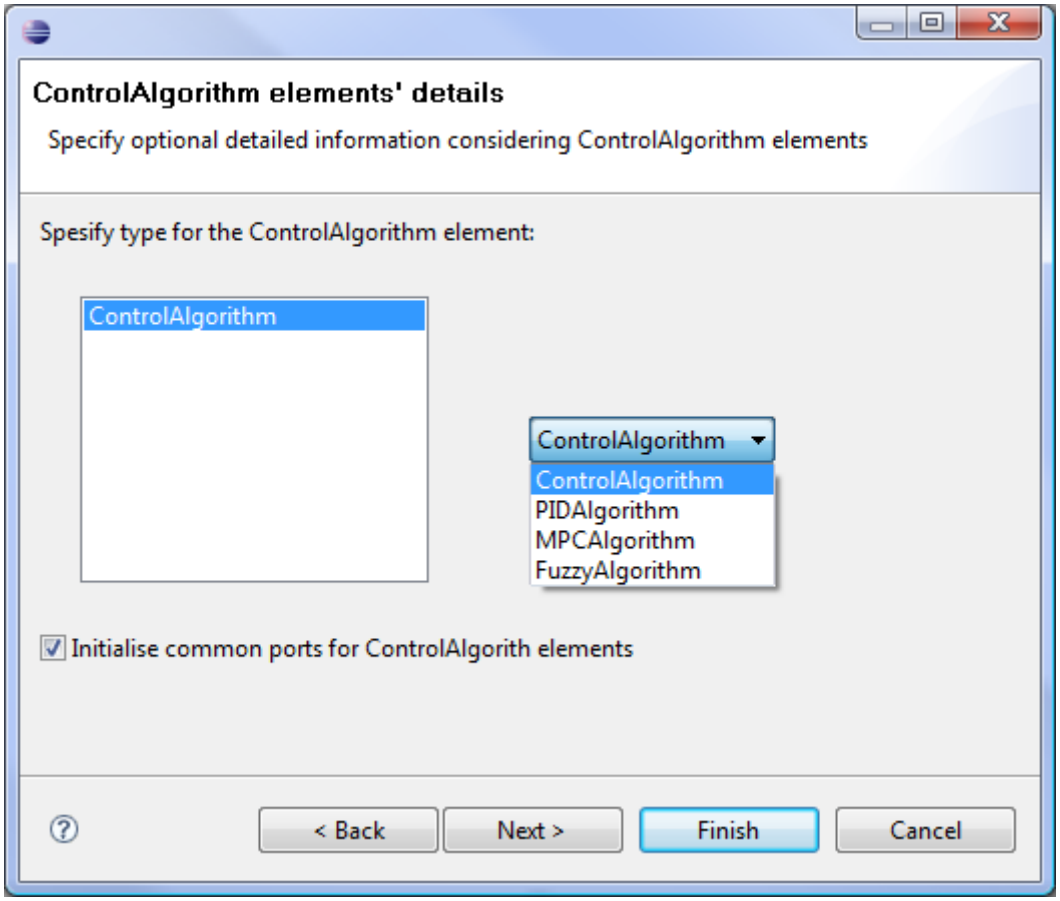
Kuva 4. *Aloitussivu asianmukaisesti täytettynä*

Käyttäjän määritettyä säätösilmukkaan haluamiensa elementtien lukumäärät, elementeille määritetään tarvittaessa lisätietoja. Kuvassa 5 on esitetty Control loop creator wizadin aloitussivua seuraava sivu, jolla käyttäjä voi asettaa kullekin mittaussisääntuloelementille (MeasurementInput) haluamansa stereotyypin wizardin tarjoamasta valikoimasta. Vasemmanpuoleisessa listassa on esitetty käyttäjän määrittämä määrä mittaussisääntuloja. Oikealla olevassa valintalaatikossa puolestaan esitetään mahdolliset stereotyypit, jotka käyttäjä voi mittaussisääntuloille asettaa. Käyttäjä voi asettaa kullekin elementille täsmälleen yhden stereotyypin. Tässä tapauksessa käyttäjä on valinnut ensimmäisen mittaussisääntulon stereotyypiksi BinaryMeasurementin ja toisen stereotyypiksi AnalogMeasurementin. Oletuksena on, että elementit alustetaan niille tyypillisillä porteilla, mutta käyttäjä voi myös estää tämän halutessaan.



Kuva 5. *Mittaussisääntulojen stereotyyppien määrittäminen*

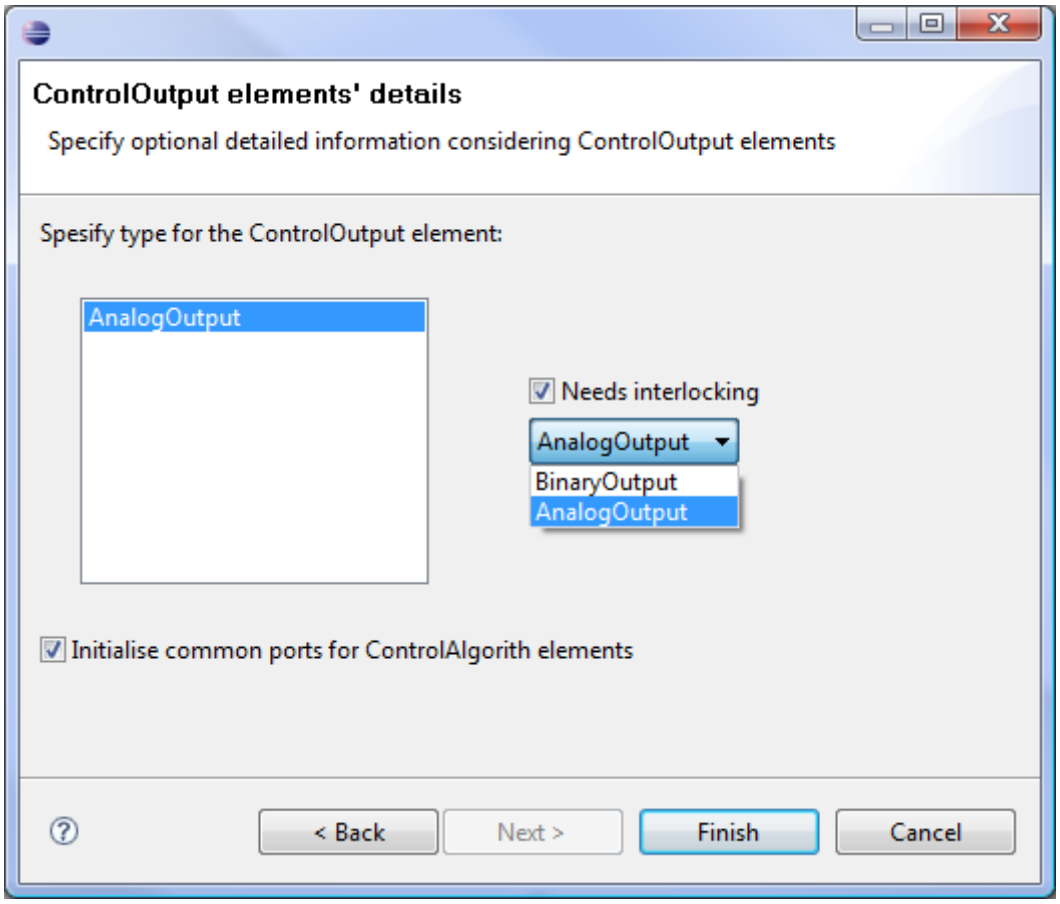
Kuvassa 6 on kuvaa 5 vastaava tilanne. Erona kuvan 5 tilanteeseen on se, että kuvassa 6 käyttäjä määrittelee stereotyypit säätöalgoritmielementeille (ControlAlgorithm), joiden vaihtoehtona ovat ControlAlgorithm, PIDAlgorithm, MPCAlgorith ja FuzzyAlgorithm. Tässä tapauksessa käyttäjä on päätynyt käyttämään yleistä ControlAlgorithm stereotyyppiä.



Kuva 6. *Säätöalgoritmien stereotyyppien määrittäminen*

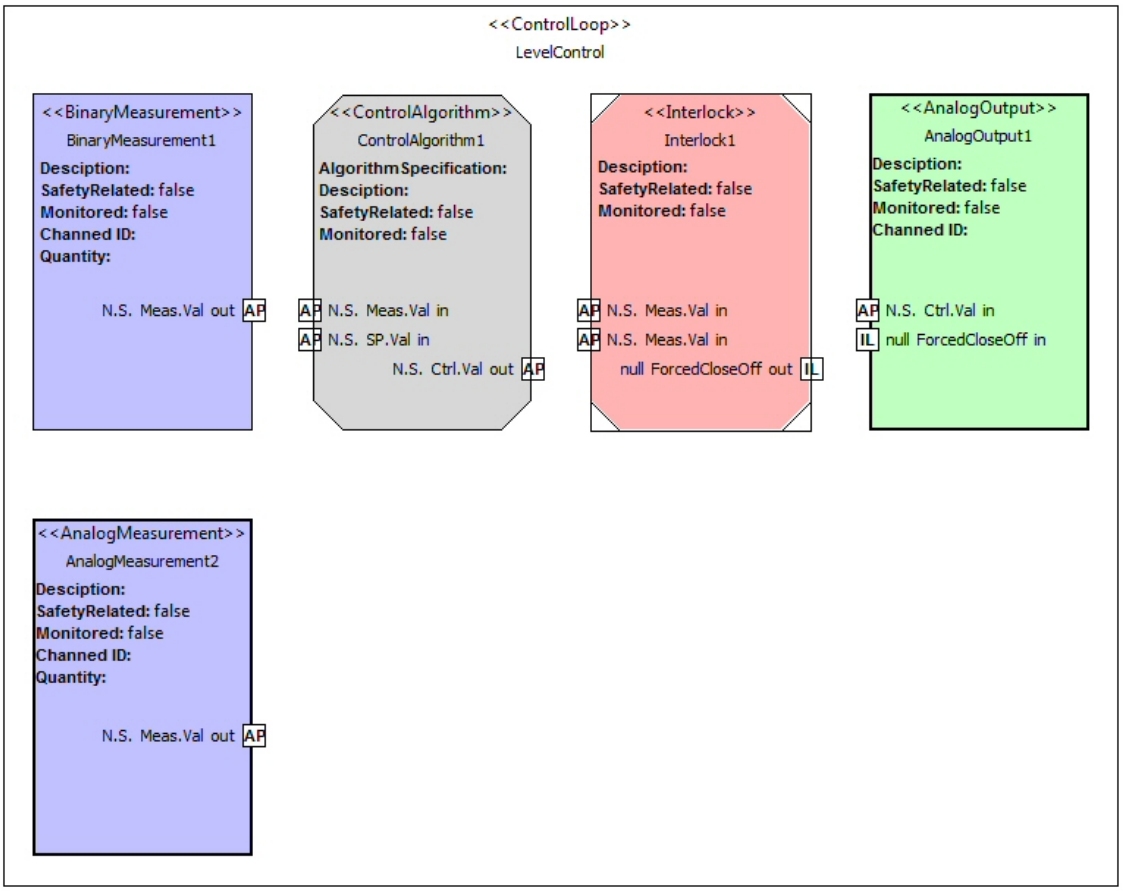
Kuvassa 7 on jälleen kuvaa 5 vastaava tilanne. Erona kuvan 5 tilanteeseen on se, että kuvassa 7 käyttäjä määrittelee stereotyypit ohjausulostuloelementeille (ControlOutput), joiden vaihtoehtona ovat BinaryOutput ja AnalogOutput. Tässä tapauksessa käyttäjä on päätynyt käyttämään AnalogOutput-stereotyyppiä. Stereotyypin lisäksi käyttäjä voi lisätä kullekin ohjauslähdölle lukituksen. Kuten liitteen alussa mainittiin, pinnankorkeuden säätö vaatii lukituksen, joka estää pinnankorkeuden ajautumisen liian korkeaan arvoon. Lukitus lisätään valitsemalla 'Needs interlocking', jolloin kyseiselle ohjauslähdölle generoituu myös lukitustoiminto.

Kuvasta voidaan havaita, että sivun Next-valinta ei ole käytettävissä. Tämä johtuu siitä, että kyseessä on wizardin viimeinen sivu. Tässä vaiheessa käyttäjä voi joko käynnistää lopputulosten generoinnin valitsemalla Finish tai palata muokkaamaan valintojaan edellisillä sivuilla valitsemalla Back. Tässä esimerkkitapauksessa käyttäjä on kuitenkin tyytyväinen tekemiinsä valintoihin joten hän siirtyy lopputulosten generointiin.



Kuva 7. *Ohjausulostulojen stereotyyppien määrittäminen*

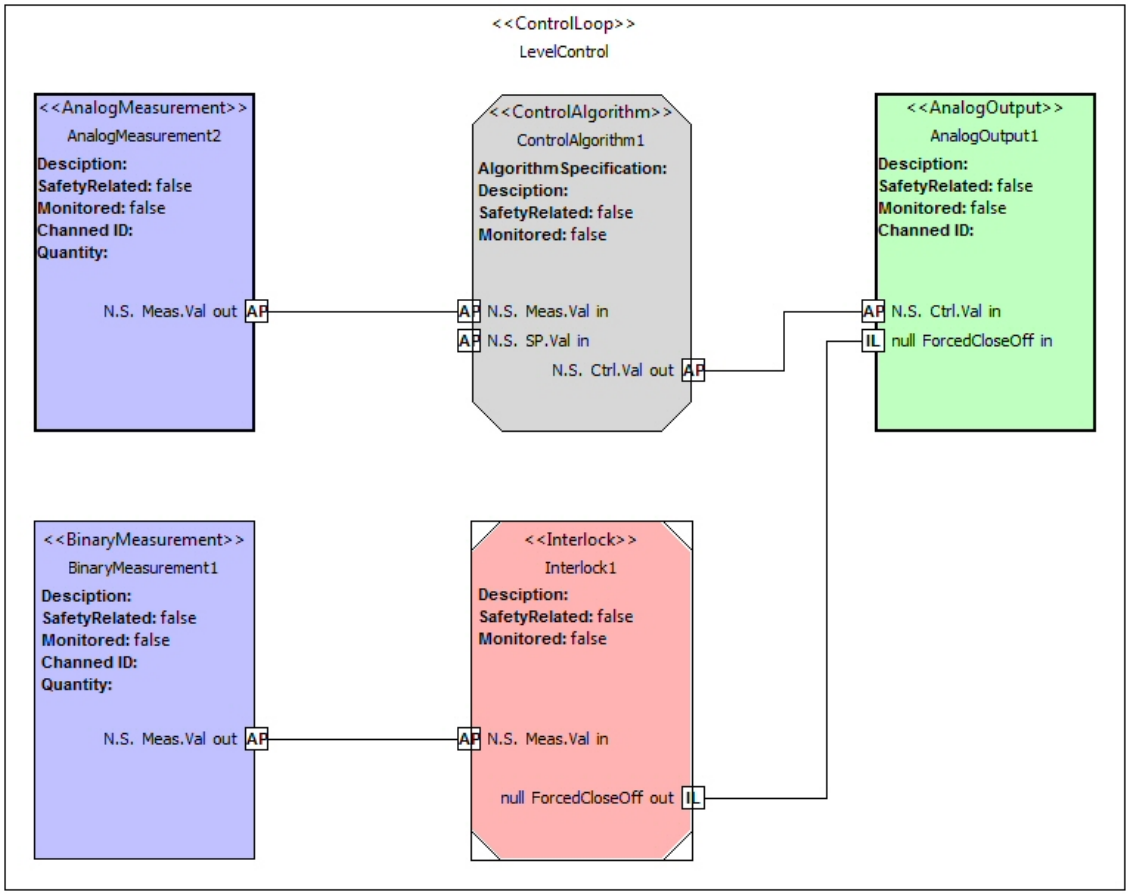
Kuvassa 8 on esitetty edellä esitetyn työnkulun lopputulokset, eli käyttäjän syötteiden perusteella generoitu säätösilmukka. Silmukka koostuu kahdesta mittaussisääntulosta sekä yhdestä säätöalgoritmista, lukituksesta ja ohjauslähdöstä. Kullekin elementille on lisäksi generoitu oletusportit. Mittaussisääntulolle generoitu portti välittää mittausarvoa ulospäin. Säätöalgoritmille generoituu kolme porttia, joista ensimmäinen ottaa vastaan mittausarvon, toinen asetusarvon ja kolmas lähettää ohjaussignaalia ulospäin. Lukituksella on kaksi mittaussisääntuloporttia sekä yksi lähtöportti, jonka tila muodostuu lukituksen logiikan ja sisääntuloporttien tilan perusteella. Ohjauslähdölle puolestaan generoituu portti, joka ottaa vastaan ohjaussignaalia ja lukitussignaalia, joka pakottaa ohjauksen kiinni.



Kuva 8. *Wizardin lopputuloksena generoitunut säätösilmukka*

Kuvassa 8 esitettiin generoitu säätösilmukka sellaisena, kuin Control loop creator wizard se toteuttaa. Säätösilmukka ei kuitenkaan tällaisenaan ole täydellinen vaan käyttäjän tulee viimeistellä silmukka. Käyttäjän tulee (tarvittaessa) järjestellä luodut elementit sopivaan järjestykseen, muokata elementtejä ja luoda porttien väliset yhteydet. Periaatteessa olisi mahdollista toteuttaa wizard, joka tiedustelee mainittujen tehtävien suorittamiseen tarvittavat tiedot käyttäjältä, mutta tämä olisi hankalaa käyttäjän kannalta, koska kokonaiskuvan hahmottamisesta tulisi vaikeaa.

Kuvassa 9 on esitetty käyttäjän täydentämä ja muokkaama lopullinen säätösilmukka pinnankorkeuden hallintaan. Käyttäjä on yhdistänyt analogisen mittauksen tuottamaan mittaussignaalia säätöalgoritmille, joka puolestaan tuottaa säätösignaalin ja lähettää sen ohjauslähtöön. Binäärinen mittaus, joka tässä edustaa tankin ylitäyttö kytkintä, puolestaan tuottaa mittausinformaatiota lukitustoiminnolle, joka lukitsee tarvittaessa ohjauslähdön. Käyttäjä on poistanut lukituselementistä toisen sisääntuloportin, jolle ei ole käyttöä. Näin ollen lukitustoiminnon looginen toteutus on triviaali, sen toimiessa muuntimena, joka muuttaa binäärimittauksen signaalin lukitussignaaliksi.



Kuva 9. *Käyttäjän täydentämä lopullinen säätösilmukka*

Vaikka wizard ei toteutakaan koko säätösilmukkaa aivan viimeiseen asti, se tarjoaa silti paljon apua säätösilmukkaa toteuttavalle käyttäjälle. Kukin elementeistä tulisi yksi-

tellen lisätä kaavioon käsin, minkä lisäksi esimerkiksi kaikille porteille tulisi määritellä sopivat määreet. Wizardin avulla kullekin elementille saadaan valmiiksi oletusportit, joita käyttäjä voi tarpeen tullen muokata. Porttien oletusarvot on pyritty valitsemaan niin, että ne vastaisivat mahdollisimman hyvin yleisesti käytettyjä tyyppejä. Esimerkiksi ohjauslähtöelementille generoituva portti, joka ottaa sisään ohjausarvon, on varsin tyypillinen kaikille ohjauslähtöelementeille.